循环结构

while 语句

计算机中有一个很经典的问题：如何用计算机求解 `1 + 2 + ... + 100`？可能可以直接用公式算，但是如果一定要让计算机依次累加来计算的话，是不是得写一串很长的加法式子？自然不用。C语言中提供了**循环**的实现方式，即我们只需要让一个变量从 1 循环自增直到 100，将中间的每个数字都累加起来，就可以得到正确结果，而 `while` 就是实现循环的三种方式之一。

`while` 的格式如下：

```
while (条件 A) {
    ......
}
```

可以看到，`while` 的格式非常简洁，并且跟 `if` 语句十分相像——只要条件 `A` 成立，就反复执行省略号的内容。如果不加大括号的话，`while` 循环的范围默认到 `while` 后的第一个分号为止。

以一开始的问题为例，我们可以先令 `n =1`，`sum = 0`，然后以 `n ≤ 100` 作为循环条件，每次把 `n` 加到 `sum` 上，再把 `n` 自增：

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

输出结果：

```
sum = 5050
```

另外，`while` 的条件中判断的是真假，因此在条件语句中的**小技巧**在此同样适用：

(1)如果表达式是 `!= 0`，则可以省略 `!= 0`；

(2)如果表达式为 `== 0`，则可以省略 `== 0`，并在表达式前添加感叹号 `!`。

实例如下：

```c
#include <stdio.h>
int main() {
    int n = 12345, count = 0;
    while(n) { //相当于 while(n != 0)
        count = count + n % 10;
        n = n / 10;
    }
    printf("%d\n", count);
    return 0;
}
```

输出结果：

```
15
```

上面的小程序实现了将 `n` 的每一位数字相加，即 `1 + 2 + 3 + 4 + 5 = 15`。`while` 循环中每次通过 `n % 10` 获取当前 `n` 的最低位，之后通过 `n = n / 10` 将最低位抹去。`while` 循环直到 `n` 变为 0 则停止，得到的 `count` 即为需要的结果。

do...while 语句

`do...while` 语句和 `while` 语句很像，但是它们的格式是上下颠倒的：

```c
do {
    ......
} while (条件 A);
```

`do...while` 语句会先执行省略号中的内容一次，然后才判断条件 `A` 是否成立。如果条件 `A` 成立，继续反复执行省略号的内容；直到某一次条件 `A` 不再成立，则退出循环。

还是 `1 + 2 + ... + 100` 进行求和，写法如下（注意 `while` 的末尾是有分号的）：

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

输出结果：

```
5050
```

这样看来，`while` 和 `do...while` 是不是一样的呢，因为上面的例子看上去连循环条件都一样？其实不是的，`do...while` 语句和 `while` 不同的一点是 `do...while` 会先执行循环体一次，然后才去判断循环条件是否为真，这就使得 `do...while` 语句的实用性远不如 `while`，因为我们碰到的大部分情况都需要能处理在某些数据下不允许进入循环的情况，例如下面这个例子：

```
#include <stdio.h>
int main() {
    int n; scanf("%d", &n);
    do {
        printf("1");
        n--;
    } while(n > 0);
    return 0;
}
```

在这个例子中，我们需要实现这样一个功能：对输入的非负整数 `n`，输出 `n` 个 1。如果采用 `do...while` 语句的写法，当读入的 `n` 大于 0 的时候都可以很好地实现功能；但是当读入的 `n` 恰好为 0 时，理论上不应该输出，但是 `do...while` 会先执行一次循环体，然后才去判断，这就使得输出了一个 1，不符合题意。当然我们可以修改判断条件或者对 `n == 0` 进行特判，但是这对一些复杂的程序逻辑来说会增加思维难度。相比较来说，直接用 `while` 就可以更直接完成功能：

```
#include <stdio.h>
int main(){
    int n; scanf("%d", &n);
    while(n > 0){
        printf("1");
        n--;
    }
    return 0;
}
```

for 语句

`for` 语句的使用频率是三种循环语句中最高的，其普适格式如下：

```
for (表达式 A; 表达式 B; 表达式 C) {
    ......
}
```

初学者看到有三个表达式不要怕，其实这样写反而***简洁***，我们等下会说明这点。先来解释这个格式的意思：

(1)在 `for` 循环开始前，首先执行表达式 `A`。

(2)判断表达式 `B` 是否成立：若成立，执行省略号内容；否则，退出循环。

(3)在省略号内容执行完毕后，执行表达式 `C`，之后回到(2)。

为了理解上面的格式，我们举一个较为常用的特例：

```
for(循环变量赋初值; 循环条件; 循环变量改变) {
    ......
}
```

这个 `for` 循环的逻辑是：先给要循环的变量赋初值，然后反复判断循环条件是否成立：如果不成立，则退出循环；如果成立，则执行省略号的内容，执行完毕后改变循环变量的值（如加 1），并重新判断循环变量是否成立，如此反复。来看一个实例：

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

输出结果：

```
5050
```

我们把 `for` 循环提出来看：

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

这个 `for` 循环的逻辑是这样的：

(1)令 `i = 1`；

(2)判断 `i ≤ 100` 是否成立：成立，令 `sum = sum + 1`，并在之后执行 `i++` 使 `i` 变为 2；

(3)判断 `i ≤ 100` 是否成立：成立，令 `sum = sum + 2`，并在之后执行 `i++` 使 `i` 变为 3；

(4)……

(5)当 `i == 100` 时，判断 `i ≤ 100` 是否成立：成立，令 `sum = sum + 100`，并在之后执行 `i++` 使 `i` 变为 101；

(6)判断 `i ≤ 100` 是否成立：发现不成立，退出循环。

于是就有 `sum = 1 + 2 + ... + 100 = 5050`。

初学者可能会认为 `for` 语句的写法有些复杂，没有 `while` 好写，其实不然，因为这三个表达式其实在 `while` 语句中全都出现了：

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

我们发现，一开始定义变量的 `i = 1` 就是在给循环变量赋初值，而 `i ≤ 100` 则是循环条件，在循环体执行完毕后的 `i++` 就是在给循环变量自增以进行下一次循环。所以 `for` 语句只是把这三个表达式都放在同一行了，这样反而可以使我们逻辑更加清晰，也更方便检查。学弟妹一定要学会写 `for` 语句，因为在大部分不太简单的题目里都需要用到。另外，`for` 语句下如果只有一个语句，则可以不加大括号，不过一般还是加上比较好，可以省去很多潜在的错误。

> 特别提醒：在 C 语言中不允许在 `for` 语句的表达式 1 里定义变量（例如 `int i` 的写法是不允许的），但是在 C++中可以，因此下面这种写法需要把文件保存为 `.cpp` 文件才能通过编译。

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

显然，随时定义临时变量才更符合我们的习惯，因此要习惯把文件保存为 `.cpp` 文件而不是 `.c` 文件，并在提交程序的时候选择 C++语言提交。由于 C++是向下兼容 C 的，C 的程序可以在 C++中运行，但是 C++中的一些好的特性（就比如刚才介绍的这点）不允许在 C 语言中运行。总而言之，在训练中，请尽量习惯让文件名的后缀为 `.cpp`。

break 和 continue

`break` 在前面讲解 `switch` 的时候已经提到过：它可以强制退出 `switch` 语句。而事实上 `break` 同样适用于循环，即在需要的场合下直接退出循环（前面介绍的三种循环语句都可以）。举一个例子：

```
#include <stdio.h>
int main(){
    int n, sum = 0;
    for(int i = 1; i <= 100; i++) {
        sum = sum + i;
        if(sum >= 2000) break;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

输出结果：

```
sum = 2016
```

上面这个代码实现了在 `1 + 2 + 3 + ... + 100` 的过程中，输出总和第一次超过 `2000` 时候的 `sum` 值，就需要在循环体中加一条 `if` 条件语句来使 `sum ≥ 2000` 的时候退出 `for` 循环。

`continue` 跟 `break` 的作用有点相似，它可以在需要的地方临时结束循环的当前轮回。实例如下：

```c
#include <stdio.h>
int main(){
    int sum = 0;
    for(int i = 1; i <= 5; i++) {
        if(i % 2 == 1) continue;
        sum = sum + i;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

输出结果：

```
sum = 6
```

在这段代码的 `for` 循环中，当满足 `i % 2 == 1`（即 `i` 为奇数）时进行 `continue`，即可将该句以下的部分直接切断不执行，执行 `i++` 后进入下一层循环。为了使 `continue` 的过程看得更清晰，我们对这段代码的执行过程进行罗列：

(1)`i == 1`：`i % 2 == 1`，因此 `continue` 执行，于是后面的语句都不执行，`i++` 后进入下层循环；

(2)`i == 2`：`i % 2 == 0`，因此 `continue` 不执行，`sum = sum + i` 得 `sum == 2`，`i++` 后进入下层循环；

(3)`i == 3`：`i % 2 == 1`，因此 `continue` 执行，于是后面的语句都不执行，`i++` 后进入下层循环；

(4)`i == 4`：`i % 2 == 0`，因此 `continue` 不执行，`sum = sum + i` 得 `sum == 6`，`i++` 后进入下层循环；

(5)`i == 5`：`i % 2 == 1`，因此 `continue` 执行，于是后面的语句都不执行，`i++` 后进入下层循环；

(6)`i == 6`：不满足 `i ≤ 5` 的条件，退出 `for` 循环。

由此，`break` 跟 `continue` 的用法都已经介绍完毕。这两个语句在编程时会非常频繁地使用，请学弟妹务必掌握它们的用法。