循环嵌套

在一个循环结构的循环体里面，又包含了一个完整的循环结构，称为嵌套循环。

```
for (外层循环变量初始化表达式; 外层循环条件表达式; 外层循环控制变量更新表达式)
{
    for (内层循环变量初始化表达式; 内层循环条件表达式; 内层循环控制变量更新表达式)
    {
        ...
    }
    ...
}

while (外层循环条件表达式)
{
    for (内层循环变量初始化表达式; 内层循环条件表达式; 内层循环控制变量更新表达式)
    {
        ...
    }
    ...
}
```

例题的代码是一个典型的 2 层嵌套循环结构；嵌套还可以是多层嵌套结构。

根据嵌套结构的性质，可以分为外层循环和内层循环。

外层循环每次都会等内层循环循环完毕，才算执行一次。

如果外层循环语句重复n次，内层循环语句重复m次，则内层循环的循环体总共会重复执行$n \times m$次。

嵌套循环可以是任意两种循环语句进行相互嵌套。

阶乘之和

【问题描述】

求 $S = 1! + 2! + 3!+....+10!$

【运行结果】

```
4037913
```

【分析】

这个问题是求 10 以内自然数的阶乘之和，可以用 for 循环来实现。

程序结构如下：

```
for(int i=1; i<=10; i++)
{
    factorial = i!;
    sum += factorial;
}
```

根据以上结构，通过 10 次的循环可以求出 1!，2!，…10!，并不断累加起来，求出 s。而求 t=i!，又可以用一个 for 循环来实现：

```
factorial = 1;
for (int j=1; j<=i; j++)
{
    factorial *= j;
}
```

【参考程序 1】

因此，完整的程序如下：

```
#include<iostream>
using namespace std;
int main()
{
    //累加
    int sum = 0;
    for(int i=1; i<=10; i++)
    {
```

```
        //累乘
        int factorial = 1;
        //求 i!
        for (int j=1; j<=i; j++)
        {
            factorial *= j;
        }
        //累加 i!
        sum += factorial;
    }
    cout << sum;
    return 0;
}
```

【参考程序 2】

实际上对于求 i!，可以根据求出的(i-1)!乘上 i 即可得到，而无需重新从 1 再累乘到 i。下面是对这个程序的优化实现：

```
#include <iostream>
using namespace std;
int main()
{
    int factorial=1, sum=0;
    for(int i=1; i<=10; i++)
    {
        //factorial 为上一个数的 i-1 的阶乘值，再乘以 i 即为 i!
        factorial *= i;
        //累加 i!
        sum += factorial;
    }
    cout << sum;
    return 0;
}
```

可以看出第二个程序的效率要比第一个高得多。第一个程序要进行$1+2+3+\cdots+10=55$次循环，而第二个程序只需要进行 10 次循环。

如果求$1!+2!+\cdots+1000!$，那么两个程序效率的区别会更加明显。以后我们会知道这两个程序的算法时间复杂度是不一样的。

阶乘之和的尾数

【问题描述】

输入n（$n \leq 10^6$），计算$S = 1! + 2! + 3! + \cdots + n!$的末 6 位（不含前导 0）。

【输入样例】

```
10
```

【输出样例】

```
37913
```

【分析】

因为只要末 6 位，所以输出时对10^6取模。

【参考程序 1】

```
#include<iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    //累加
    int sum = 0;
    for(int i=1; i<=n; i++)
    {
        //累乘
        int factorial = 1;
        //求 i!
        for (int j=1; j<=i; j++)
        {
            factorial *= j;
        }
        //累加 i!
        sum += factorial;
    }
    cout << sum % 1000000;
    return 0;
}
```

我们发现，当 n=100 时，输出-961703。乘法溢出了。

【参考程序 2】

```
#include<iostream>
#include<ctime>
using namespace std;
const int MOD = 1000000;
int main()
{
    int n;
    cin >> n;
    int sum = 0;
    for (int i=1; i<=n; i++)
    {
        int factorial = 1;
        for (int j=1; j<=i; j++)
        {
            factorial = factorial * j % MOD;
        }
        sum = (sum + factorial) % MOD;
    }
    cout << sum << endl;
    cout << "Time used=" << 1.0 * clock()/CLOCKS_PER_SEC << endl;
    return 0;
}
```

这个程序对溢出做了优化，输入 100 就没有问题了。

不过输入 100000 呢？在老师的电脑上要运行 40 多秒钟才能出来结果。

其中计时函数 clock()会返回程序目前为止运行的时间。在程序结束之前调用它，便可获得程序的运行时间。这个时间除以常数 CLOCKS_PER_SEC 得到的值就是程序运行了多少“秒”。

【参考程序 3】

```
#include<iostream>
#include<ctime>
using namespace std;
const int MOD = 1000000;
int main()
{
    int n;
```

```
    cin >> n;
    int sum = 0;
    int factorial = 1;
    for (int i=1; i<=n; i++)
    {
        factorial = factorial * i % MOD;
        sum = (sum + factorial) % MOD;
    }
    cout << sum << endl;
    cout << "Time used=" << 1.0 * clock()/CLOCKS_PER_SEC << endl;
    return 0;
}
```

我们再次优化代码，这个代码在老师的电脑上只要 2 秒钟就能出来结果了。