前言：

大家一定要积极尝试把课上学到的**知识映射到代码里**，避免知识与代码之间割裂。
希望资料里给出的一些代码和总结能够有助于大家理解和回忆。
不过，绝知此事要躬行，亲手写的bug，才更能加深大家对知识点的理解和应用。
所以希望大家：
①及时总结自己所有出现过的bug，自己作业、课堂练习等踩过的坑，尽量避免将来掉进同样的坑里；
②建议总结解决一类问题的可参考方法及注意事项，避免记住了语法但不会写代码的窘境。

1、for循环相关的基础语法：

逗号表达式的执行顺序是从A到B， C作为整个逗号表达式的结果。
【上句的ABC分别是什么？AB都是方位词】

for循环的循环变量可以在控制行之前定义，也可以在控制行的表达式1中定义。
【什么是控制行？后面有分号吗？什么是表达式1？】

如果是在控制行的表达式1中定义，那么该变量就只能在这个循环中使用。
【之后学了第6章函数，可以联系其中的“变量的定义域”的概念，会有助于加深理解】

for循环的循环体理论上应该只有一个语句（可以是空语句，即只有一个分号）。
如果循环体有多个语句，要用大括号把它们合并成一个复合语句，在语法上视为一个语句。
【什么是循环体？有分号吗？复合语句大括号外有分号吗？复合语句里每个语句后有分号吗？】

```
for(int i =0; i<10; ++i){
        int x = i+1;
        cout << x;
}
cout << x+1; //编译报错：x无定义
```

```
int k;
cin >> k;
if (k < 10)  int x = k+1;
else  int x = k-1;
cout << x; //编译报错：x无定义
```

在左上代码中，x是在for循环的循环体（复合语句）中定义的变量，因此不能在外部使用。在右上代码中，x是在then子句或else子句中定义的，虽然没有大括号，但也是程序块。
【“程序块”的概念同样会在第6章里讲解，到时候可以回顾这里的知识点】

for循环控制行的表达式1是在开始进入循环的时候执行的，**且只执行一次。**
for循环控制行的表达式2是在**每次将要进入**一个循环周期时判断真假的。
for循环控制行的表达式3是在每个循环**周期结束**时执行的。
【上一段中加粗的部分能够理解吗？可以想象一个for循环的运行过程吗？】

for循环控制行的表达式之间要用分号来隔开。
【几个表达式？几个分号？】

最简单的死循环：for (; ;);

2、使用for循环的代码中常见的问题：

计数用的变量**没有初始化为0**，导致执行结果混乱，例如出现非常大的整数。

for的循环控制行后面多加了分号，导致一个空语句成为了编译器检测到的循环体，程序就不会进入我们真正想要的循环语句。

for循环语句**没有出口**导致死循环（没有控制行的表达式2也没有break）。

for循环控制行里写了i++，还在循环内部又写了i++，导致结果错误。

for循环的循环体多于一个语句，但却**没加大括号**。

3、while和do…while循环相关的基础语法：

while循环的条件表达式相当于for循环控制行的表达式2。
【注意它**是继续循环**的条件，**不是终止循环**的条件】
每次将进入一个循环周期之前（包括第一次之前）都要判断该条件表达式的真假。

do…while循环要在执行完第一个循环周期之后，才判断条件表达式的真假。
注意do…while循环的**控制行末尾有个分号**。

【记住哪个是先判断、哪个是先运行了吗？】

4、循环的中途退出

区分break和continue：
break是**终止循环**；对于多层循环嵌套，要注意终止外层循环的条件（如下图）。
continue是**跳过**当前循环周期的**剩余部分**（for循环控制行的表达式3仍会执行）。

```
int a, b, c, d; bool flag = false;
for (a=1; a<=4; ++a) {
   for (b=1; b<=4; ++b) {
      if (a==b) continue;
      else
         for (c=1; c<=4; ++c)
            if (c==a||c==b) continue;
            else {
               d = 10 – a – b – c;
               if ((a==1)+(b==4)+(c==3)==1 && (b==1)+(a==4)+(c==2)+(d==3)==1
               && (b==4)+(a==3)==1 && (c==1)+(d==4)+(b==2)+(a==3)==1) {
                  cout << a << b << c << d;
                  flag = true; break; }
      if (flag) break; }
   if (flag) break; }
```

5、枚举法和贪婪法

枚举法的基本思路：
如果有两个约束条件，则先用循环枚举出所有符合第一个约束条件的情况，再检查各个情况是否符合第二个约束条件（如课本上的例4.13买水果问题）。
如果问题只有一个解，**找到该解后应使用break立即结束循环。**

贪婪法的基本思路：
在求解过程的每一步都选取一个局部最优的策略，把问题规模缩小，最后把每一步的结果合并起来形成一个全局解（如课本上的硬币找零问题）。

6、一些小技巧小方法小总结

用循环来获取多个变量中的**最大值**max，记得要先把max赋值为这些变量所能取到的**下限**（比如考试成绩是0-100，那么max就先赋值为0）。
如果是要获取**最小值**min，则要先把min赋值为这些变量所能取到的**上限**（比如100）。

注意各类运算符的**优先级**会影响运算顺序，不能想当然。
最最安全的方法就是**加小括号**。

做整型数的乘方运算可以用循环来实现，而不能用pow函数。
因为pow函数会把整型数转换为浮点数来计算，导致误差。
且该误差是在pow函数内部产生的，**无法通过将它的返回值转换回整型来消除**。

先后使用**两次循环**时，一些在外部定义的变量可能需要在开始**第二次循环前重新赋值**。
【例如用两次二分查找分别找出一个数列中相同的数的左右边界，中间要重置low和high】

在**循环嵌套**的代码中有个典型错误，即在内层循环开始前，**某些变量忘记重置**，导致这些变量仍然保留上次内层循环后的结果。
如以下代码中，内层while循环开始前忘了重置j。

```
int i=0, j=0, a[10][10] = {初始化列表内容略};
while(i<10){
      while(j<10){
          cout << a[i][j];
          ++j;
      }
      ++i;
}
```