4. 技能 1 ——程序调错

授课教师：游伟 副教授、孙亚辉 副教授

授课时间：周二14:00 – 15:30，周四10:00 – 11:30（教学三楼3304）

上机时间：周二18:00 – 21:00（理工配楼二层机房）

课程主页：https://www.youwei.site/course/programming

目录

4.1 常见错误类型

- 编译错误
- 链接错误
- 运行时错误
- 逻辑错误

例子：YOJ 7. 求三角形面积

题目描述

定三角形三条边的长度，求此三角形的面积。

提示：海伦公式 $S=\sqrt{q\times(q-a)\times(q-b)\times(q-c)}$，其中 $q=(a+b+c)/2$。

输入格式

三个用空格隔开的小于1000的正整数。（输入保证这三个数能够构成三角形）

输出格式

输出一个数，为所求的三角形面积，四舍五入保留2位小数。

输入样例

3 4 5

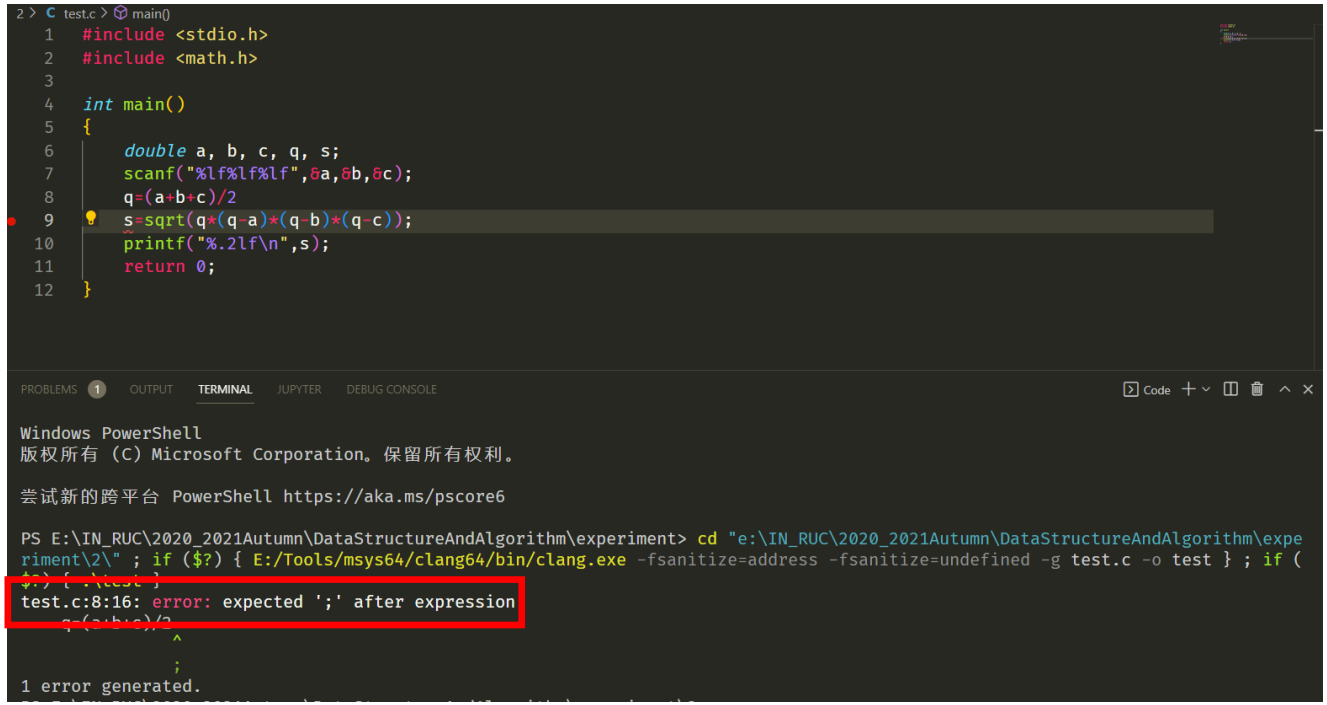
输出样例

6.00

编译错误（Compile Error）

- 编译错误发生时，表示程序中存在语法错误，编译器无法根据约定的语法对程序进行编译。
- 编译器会输出错误发生的地方，一般的形式为（不同编译器不同）：

文件名：行号：列号。



链接错误（YOJ上一般不会发生）

- 在一个程序中需要使用其他文件中的程序，而没有在编译的时候指定时，编译器找不到对应的代码，会发生链接错误。
- 错误发生时，编译器也会提示文件、行号、列号等相关信息。

```c
/* Linking error */

#include <stdio.h>
#include <math.h>

/* 使用外部的代码，用于计算 (a + b + c) / 2 的结果 */
extern double func(double a, double b, double c);

int main () {
    double a, b, c, q, s;
    scanf ("%lf%lf%lf", &a, &b, &c);
    q = func(a, b, c);
    s = sqrt(q * (q - a) * (q - b) * (q - c));
    printf ("%.2lf\n", s);
    return 0;
}
```

```
/usr/bin/ld: /tmp/b-223cb4.o: in function `main':
/[redacted]/code/b.c:12: undefined reference to `func'
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

运行时错误（Runtime Error）

- 在程序运行中，发生一些问题导致程序崩溃，包括但不限于：
  - 除法中，除数为0
  - 数组越界（不用编译器行为不一样，也可能导致逻辑错误，也可能不会发生错误）
  - 访问空指针
  - 访问非法内存

```c
#include <stdio.h>
int main()
{

    int i, j;
    printf("please input: ");
    scanf("%d", &i);
    j = 10 / i;
    printf("%d\n", j);
}
```

```
please input: 0
Floating point exception (core dumped)
```

Sanitizer：错误类型+错误发生的位置

```c
/* Runtime Error */

#include <stdio.h>
#include <math.h>

int main () {
    double a[3], q, s;
    scanf ("%lf%lf%lf", &a[1], &a[2], &a[3]);
    q = (a[1] + a[2] + a[3]) / 2;
    s = sqrt(q * (q - a[1]) * (q - a[2]) * (q - a[3]));
    printf ("%.2lf\n", s);
    return 0;
}
```

```
==40==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffefafdd718 at pc 0x000000438061 bp 0x7ffefafdd5a0 sp 0x7ffefafdcd20
WRITE of size 8 at 0x7ffefafdd718 thread T0
    #0 0x438060 in scanf_common(void*, int, bool, char const*, __va_list_tag*) (/mnt/e/OneDrive - ruc.edu.cn/朋/RUC/2022-2023秋季学期/程设助教/debug教程/lesson1/code/c+0x438060)
    #1 0x438eab in __isoc99_scanf (/mnt/e/OneDrive - ruc.edu.cn/朋/RUC/2022-2023秋季学期/程设助教/debug教程/lesson1/code/c+0x438eab)
    #2 0x4c341b in main /mnt/e/OneDrive - ruc.edu.cn/朋/RUC/2022-2023秋季学期/程设助教/debug教程/lesson1/code/c.c:8:5
    #3 0x7f67ba6af082 in __libc_start_main /build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:308:16
    #4 0x41b2ed in _start (/mnt/e/OneDrive - ruc.edu.cn/朋/RUC/2022-2023秋季学期/程设助教/debug教程/lesson1/code/c+0x41b2ed)
```

逻辑错误（Wrong Answer）

- 程序中因为部分细节没有考虑清楚，导致一些测试点的输出与预期不符。
- 例题中，题干说明了输入为3个**正整数**，但是如果以int类型存储输入的数，可以通过样例，但是无法通过所有测试点（为什么）。

```c
/* Wrong Answer */

#include <stdio.h>
#include <math.h>

int main () {
    int a, b, c;
    double q, s;
    scanf ("%d%d%d", &a, &b, &c);
    q = (a + b + c) / 2;
    s = sqrt(q * (q - a) * (q - b) * (q - c));
    printf ("%.2lf\n", s);
    return 0;
}
```

资源超过限制

- 程序运行时间超过限制
- 程序使用内存大小超过限制

程序超时（Time Limit Exceeded）

- 程序运行的时间超过题目的时间限制，会出现程序超时的问题。
- 可能的原因有：
  - 程序因为各种原因（运行时错误、逻辑错误、手抖打错等）出现死循环
  - 程序效率过低，这方面设计程序的时间复杂度

```c
/* Time Limit Exceeded */

#include <stdio.h>
#include <math.h>

int main () {
    double a, b, c, q, s;
    scanf ("%lf%lf%lf", &a, &b, &c);
    q = (a + b + c) / 2;
    s = sqrt(q * (q - a) * (q - b) * (q - c));
    printf ("%.2lf\n", s);
    while (1);
    return 0;
}
```

程序超内存（Memory Limit Exceeded）

- 程序使用的内存大小超过了题目的限制。
- 由于YOJ的内存限制较大，一般不会出现这种情况。

```
1   /* Memory Limit Exceeded */
2
3   #include <stdio.h>
4   #include <math.h>
5
6   double a[80000000], q, s;
7
8   int main () {
9       for (int i = 1; i < 80000000; ++i) a[i] = 2147483647;
10      scanf ("%lf%lf%lf", &a[1], &a[2], &a[3]);
11      q = (a[1] + a[2] + a[3]) / 2;
12      s = sqrt(q * (q - a[1]) * (q - a[2]) * (q - a[3]));
13      printf ("%.2lf\n", s);
14      return 0;
15  }
```

4.2 程序调试基本方法

- 使用printf进行调试
  - 对代码进行“粗筛”，初步判断错误的位置
  - 对代码进行“细察”，确定错误的位置
- 使用调试工具进行调试
  - 设置断点
  - 单步执行
  - 查看变量

使用printf进行调试

- 存在错误的代码：

C printf.c > calc_s(double, double, double, double)

```c
#include <stdio.h>
#include <math.h>

double calc_s(double q, double a, double b, double c) {
    double q1 = q - a;
    double q2 = q + b;
    double q3 = q - c;
    return sqrt(q * q1 * q2 * q3);
}

int main () {
    double a, b, c, q, s;
    scanf ("%lf%lf%lf", &a, &b, &c);
    q = (a + b + c) / 3;
    s = calc_s(q, a, b, c);
    printf ("%.2lf\n", s);
    return 0;
}
```

使用printf进行调试

- Step 1：对代码进行“粗筛”，初步判断错误的位置

printf.c > main()

```c
#include <stdio.h>
#include <math.h>

double calc_s(double q, double a, double b, double c) {
    double q1 = q - a;
    double q2 = q + b;
    double q3 = q - c;
    return sqrt(q * q1 * q2 * q3);
}

int main () {
    double a, b, c, q, s;
    scanf ("%lf%lf%lf", &a, &b, &c);
    q = (a + b + c) / 3;

    printf ("q = %.2lf\n", q);

    s = calc_s(q, a, b, c);
    printf ("%.2lf\n", s);
    return 0;
}
```

```
3 4 5
q = 4.00
-nan
```

使用printf进行调试

- Step 2：对代码进行“细察”，确定错误的位置

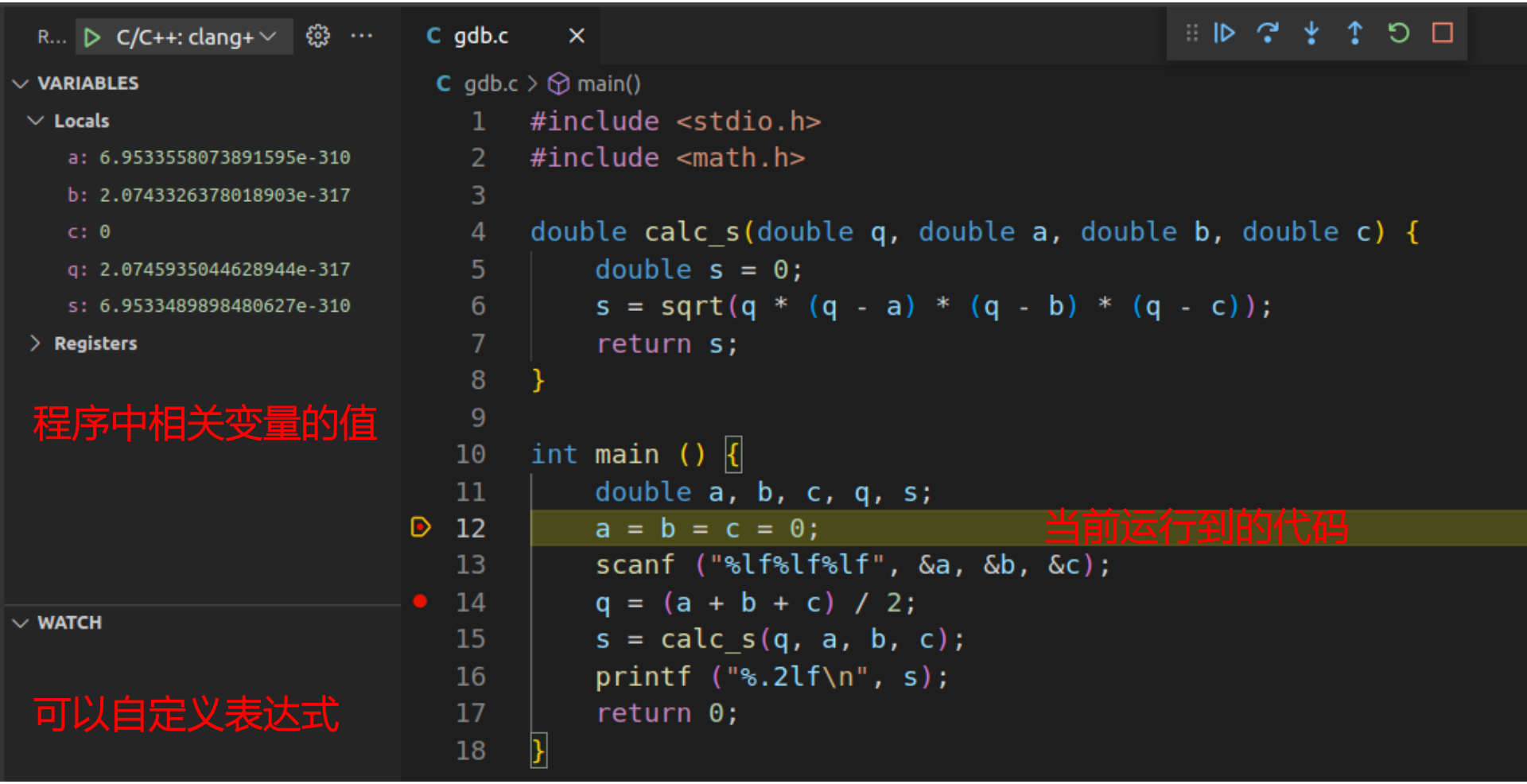
```
4   double calc_s(double q, double a, double b, double c) {
5       double q1 = q - a;
6       printf ("q1 = %.2lf\n", q1);
7       double q2 = q + b;
8       printf ("q2 = %.2lf\n", q2);
9       double q3 = q - c;
10      printf ("q3 = %.2lf\n", q3);
11      return sqrt(q * q1 * q2 * q3);
12  }
13
14  int main () {
15      double a, b, c, q, s;
16      scanf ("%lf%lf%lf", &a, &b, &c);
17      q = (a + b + c) / 2;
18      printf ("q = %.2lf\n", q);
19      s = calc_s(q, a, b, c);
20      printf ("%.2lf\n", s);
21      return 0;
22  }
```

```
3 4 5
q = 6.00
q1 = 3.00
q2 = 10.00
q3 = 1.00
13.42
```

使用调试工具进行调试

- 使用vscode中的图形化界面辅助调试

调试功能键，从左到右为continue, step over, step into, step out, restart, stop



使用调试工具进行调试

- Step 1：设置断点

C gdb.c > main()

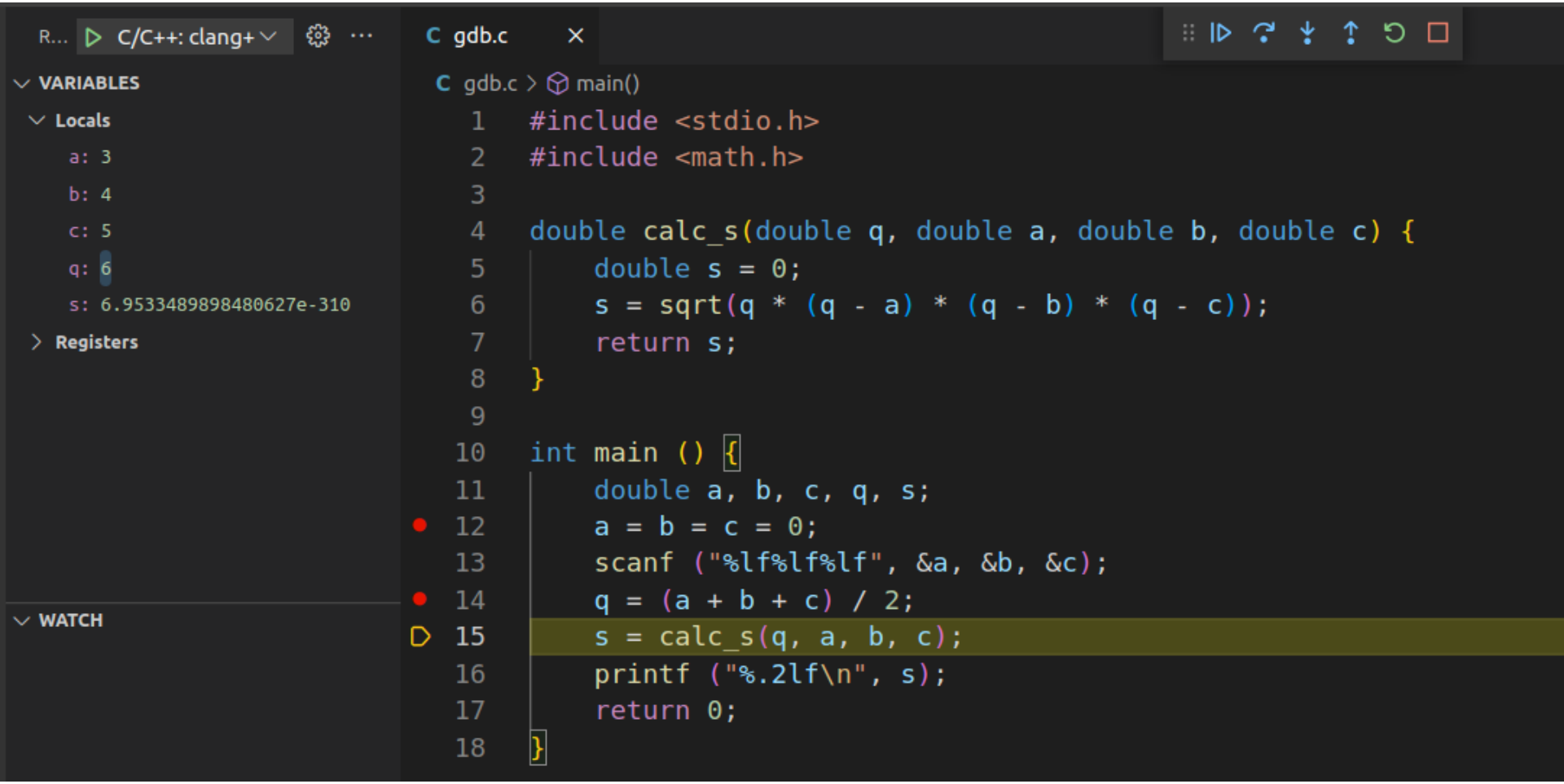
```c
#include <stdio.h>
#include <math.h>

double calc_s(double q, double a, double b, double c) {
    double s = 0;
    s = sqrt(q * (q - a) * (q - b) * (q - c));
    return s;
}

int main () {
    double a, b, c, q, s;
    a = b = c = 0;
    scanf ("%lf%lf%lf", &a, &b, &c);
    q = (a + b + c) / 2;
    s = calc_s(q, a, b, c);
    printf ("%.2lf\n", s);
    return 0;
}
```

使用调试工具进行调试

- Step 2：调试程序
  - continue：执行到下一个断点
  - step over：执行下一行程序
  - step into：执行下一步程序
  - step out：结束当前的函数
  - restart：重新开始
  - stop：结束

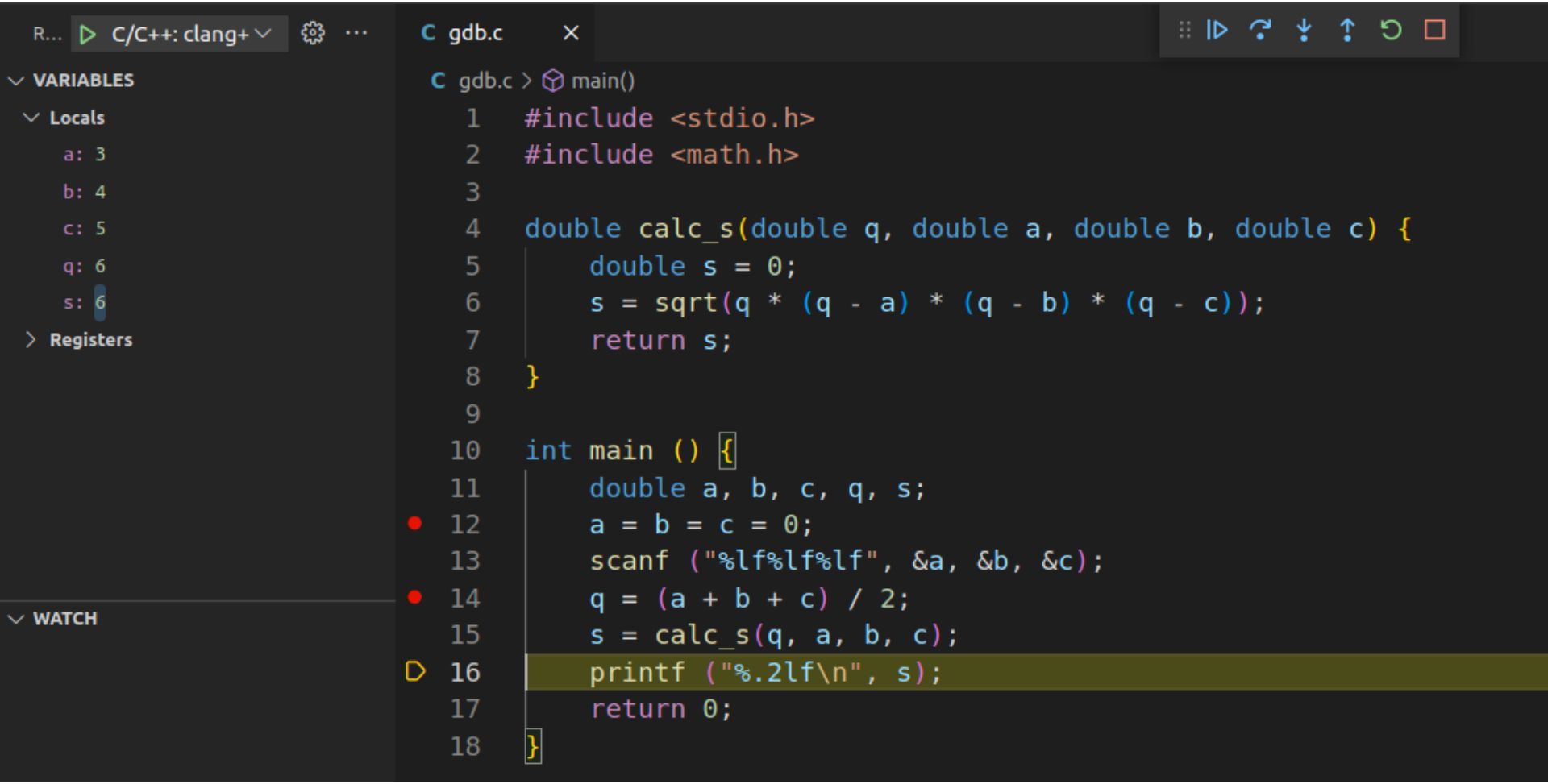
调式工具：step over

- 执行step over之前：



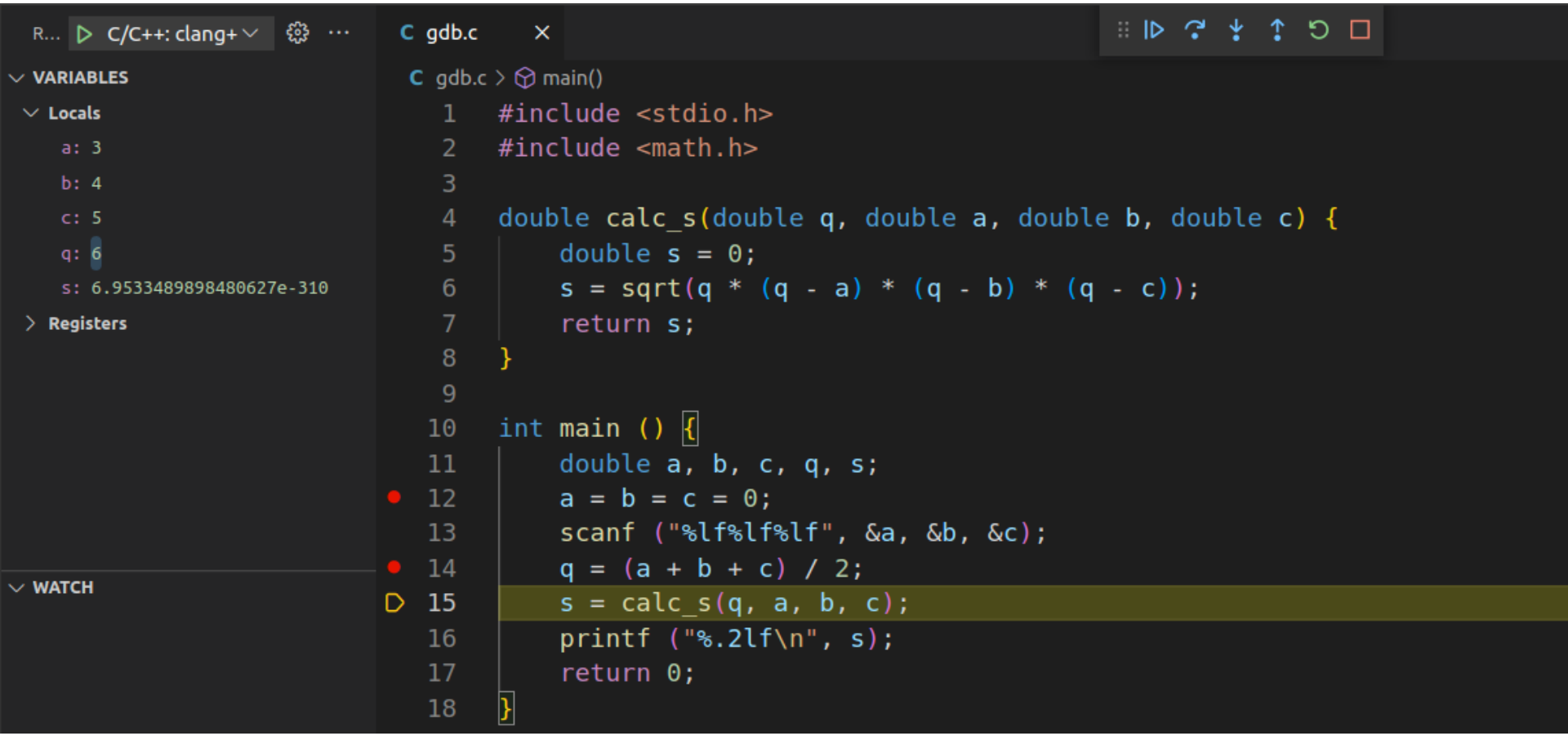
调式工具：step over

- 执行step over之后：



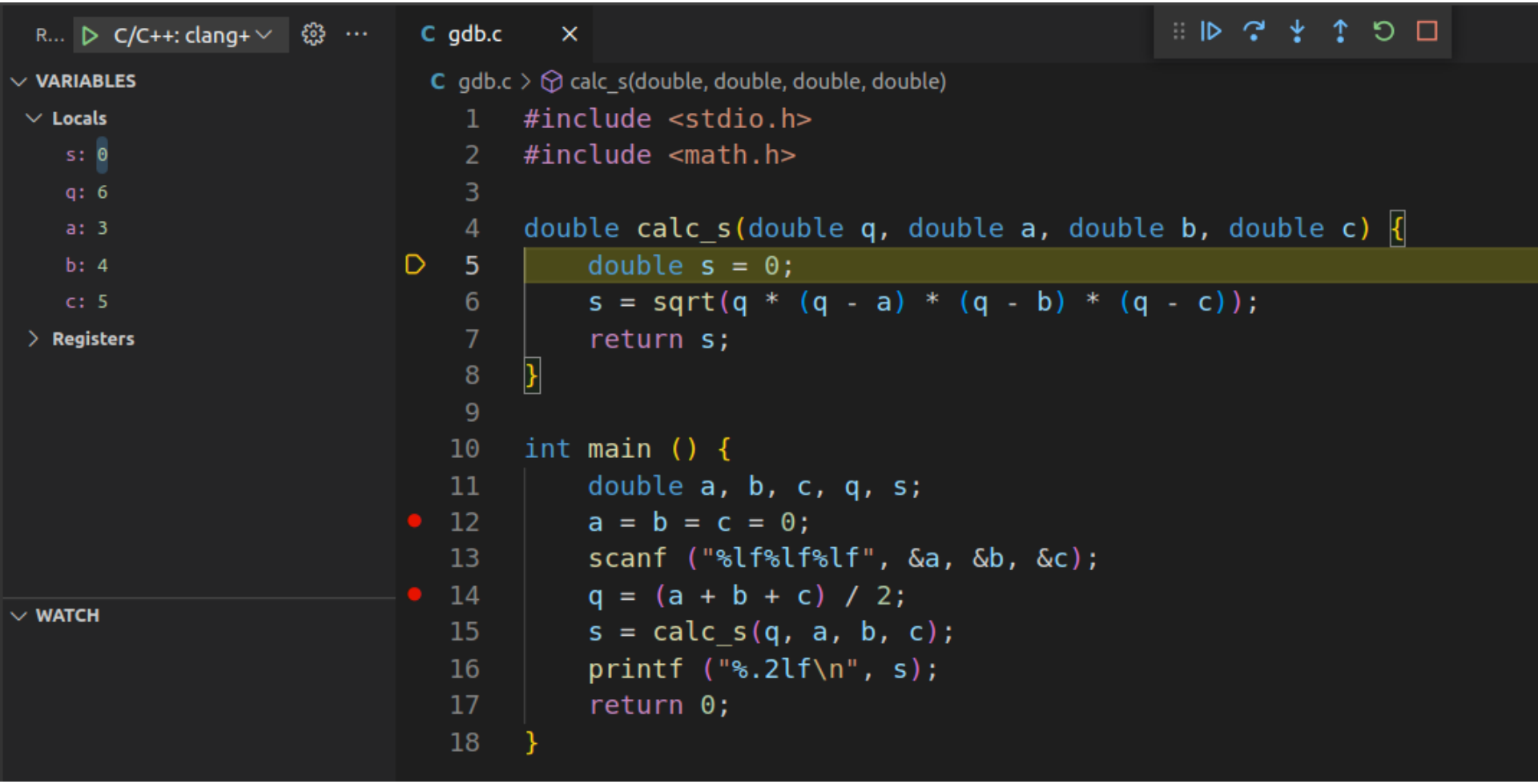
调试工具：step into

- 执行step into之前：



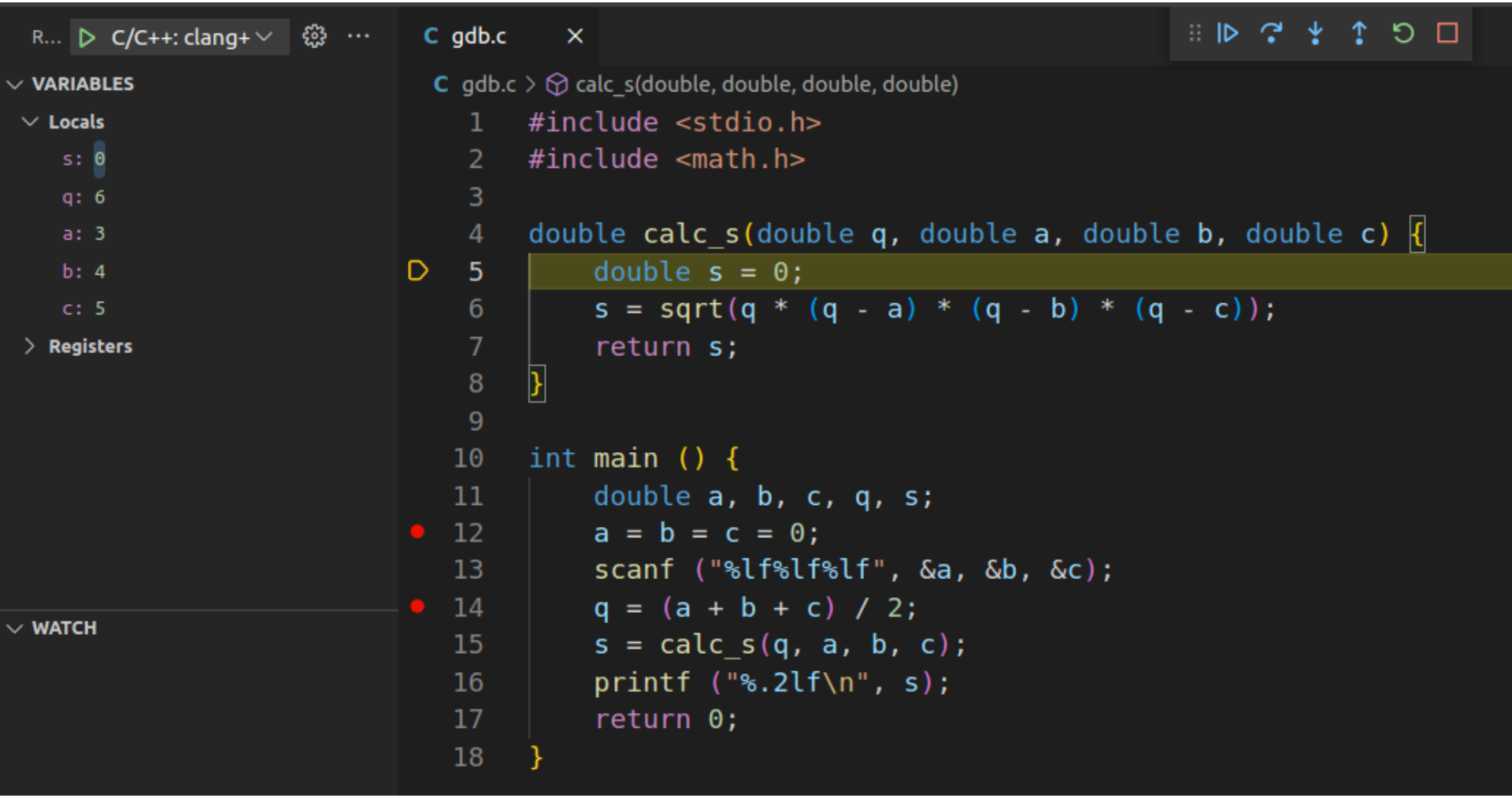
调试工具：step into

- 执行step into之后：



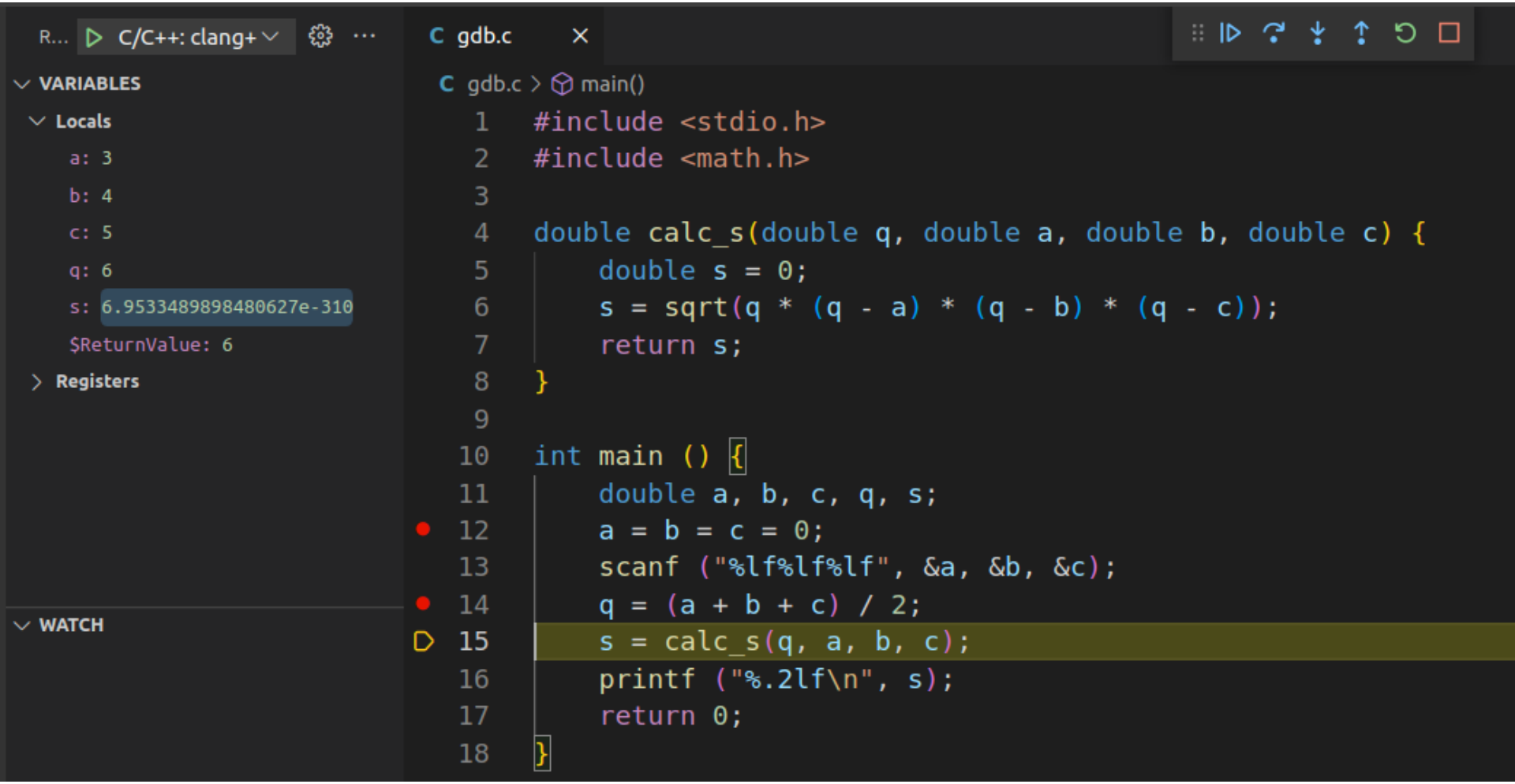
调试工具：step out

- 执行step out之前：



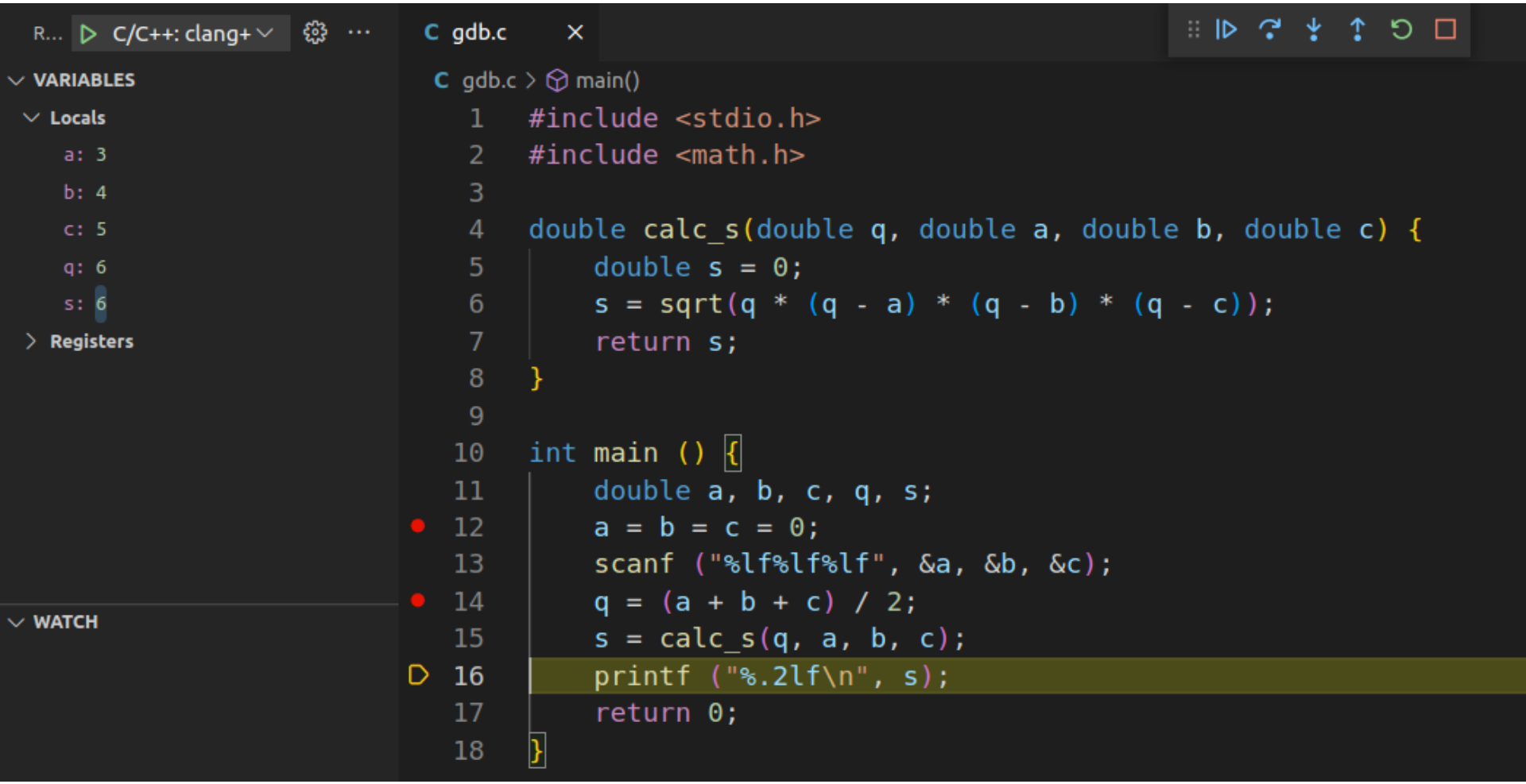
调试工具：step out

- 执行step out之后：



调试工具：step out

- 再执行一次step into



调试练习-1

- YOJ-183：一元三次方程求解（演示）
- YOJ-8：计算学分绩点（练习）
- YOJ-448：按要求读入并输出数据（练习）
- YOJ-298：被整除的数（练习）

4.3 分支循环结构的调错

- 代码风格规范
- 条件断点设置
- 断点位置设置经验

代码风格规范

- 规范使用缩进容易看清程序的逻辑关系，降低出现bug的概率
  - 子作用域进行缩进，能够看清层次关系
  - 运算符之间加空格，能够理清式子脉络

```
scanf("%lf%lf",&a,&b);
for(;a>20;){
   a=a-e; }
c=a;
for(;1;i++){
sum+=c;
c*=(-1)*a*a;
c/=(2*i*(2*i+1));
if(fabs(c)<b){
break;
}
}
printf("%lf\n",sum);
```

```
scanf("%lf%lf", &a, &b);
for (; a > 20;)
{
    a = a - e;
}
c = a;
for (; 1; i++)
{
    sum += c;
    c *= (-1) * a * a;
    c /= (2 * i * (2 * i + 1));
    if (fabs(c) < b)
    {
        break;
    }
}
printf("%lf\n", sum);
```

条件断点设置

- 当循环次数较多时，在每次循环中设置断点，会导致断点触发次数多，反复查看正确的过程，浪费时间
- 先判断错误可能发生的位置，使用if语句在循环中设置相应的“无作用”语句，将断点设置在这个语句上
- 下面以 YOJ-148. 傻大木买军火 为例

输入文件（data6.in）

```
60
```

输出文件（data6.out）

```
10 2 28
10 3 18
12 1 28
12 3 18
14 2 18
16 1 18
```

选手输出

```
10 1 28
10 2 28
10 3 18
12 1 28
12 2 18
12 3 18
14 2 18
16 1 18
```

条件断点设置

```
for(x=0;x<=n/2;x+=2)
{
    for(y=1;y<n/6;y++)
    {
        for(z=1;z<=n-8;z++)
        {
            if
            (
                (x!=y&&x!=z&&y!=z)
                &&((x>=y&&x<=z)||(x>=z&&x<=y))
                &&(2*x+6*y+z>=n*0.9&&2*x+6*y+z<=n)
                &&(z%10==8||z/10%10==8)
                &&((y<10&&x>=10)||(y>=10))
            )
            {
                printf("%d %d %d\n",x,y,z);
                flag=1;
            }
        }
    }
}
```

```
for(x=0;x<=n/2;x+=2)
{
    for(y=1;y<n/6;y++)
    {
        for(z=1;z<=n-8;z++)
        {
            if (x == 10 && y == 1 && z == 28)
                printf(' '); // “无作用”语句
            if
            (
                (x!=y&&x!=z&&y!=z)
                &&((x>=y&&x<=z)||(x>=z&&x<=y))
                &&(2*x+6*y+z>=n*0.9&&2*x+6*y+z<=n)
                &&(z%10==8||z/10%10==8)
                &&((y<10&&x>=10)||(y>=10))
            )
            {
                printf("%d %d %d\n",x,y,z);
                flag=1;
            }
        }
    }
}
```

断点位置设置经验

- 穷举类问题
  - 多输出答案：断点设置在输出答案的位置
  - 少数出答案：根据缺少的答案设置条件断点
- 死循环类问题
  - 在程序中每个循环结束的位置设置断点，确定出现问题的循环
  - 在循环中的任意位置设置断点，注意查看循环变量是否正确修改
  - 在循环边界的位置设置条件断点（如：i <= n，在n-1的位置设置断点），查看循环能否正确到达边界，边界是否正确处理
- 其他情况
  - 根据错误，猜测出现错误的原因（如：没有取负数？精度误差？）
  - “粗筛”+“细察”

循环中的循环变量

- 要注意循环变量是否被无故修改
  - 被修改可能导致程序出现死循环
  - 被修改可能导致循环内部的其他操作出现错误
- 尽量不要直接使用或者修改循环变量
  - 因为你的程序可能会直接或者间接修改该循环变量的值，往往不容易被发现
- 以yoj-414 水仙花数 为例

循环中的循环变量

- 如果想在循环中使用循环变量，可以将其赋值给另一个变量
  - 循环变量i在循环体内部被修改导致错误
  - 可以将i赋值给j，用j去进行计算

```
for(int i = a; i < e + 1; i++) {
    b = i % 10;
    i = i / 10;
    c = i % 10;
    i = i / 10;
    d = i % 10;
    if(i == pow(b, 3) + pow(c, 3) + pow(d, 3))
    {
        printf("%d ", i);
    }
}
```

```
for(int i = a; i < e + 1; i++) {
    int j = i;
    b = j % 10;
    j = j / 10;
    c = j % 10;
    j = j / 10;
    d = j % 10;
    if(j == pow(b, 3) + pow(c, 3) + pow(d, 3))
    {
        printf("%d ", j);
    }
}
```

暴力枚举

- 在做枚举时要分析清楚多重循环
  - 要注意循环的重数，以及明确对应变量的意义
- 每次枚举的变量要记得“复原”
  - 在枚举完上一个情况后，要记得恢复某些每次枚举都有用到的变量，否则此次枚举的初始情况便于上一次不同，会导致潜在的错误
  - 在恢复时要考虑好所有情况，尤其是循环末尾带有if判断语句，往往会在某一个条件处遗漏“复原”操作
- 以yoj-414 水仙花数 为例

暴力枚举

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int n, m, s = 0;
    scanf("%d %d", &n, &m);
    for (int i = n; i <= m; i++)
    {
        for (int j = i; j > 0; j /= 10)
        {
            s += pow((j % 10), 3);
        }
        if (s == i)
        {
            printf("%d ", i);
            s = 0;
        }
        else
            s = 0;
    }
    return 0;
}
```

- 这两处都要保证s被重置为0

调试练习-2

- YOJ-636：博饼游戏（演示）
- YOJ-414：水仙花数（演示）
- YOJ-180-1：计算cos的近似值（练习）
- YOJ-180-2：计算cos的近似值（练习）

- YOJ-148-1：傻大木买军火（演示）
- YOJ-414：水仙花数（演示）
- YOJ-148-2：傻大木买军火（练习）
- YOJ-147：教室排课（练习）

4.4 数组的调错

- 出错类型总结
- 例子：YOJ-291

出错类型总结

- 数组越界
  - 忽略字符串结尾的"\0"字符
  - 创建长度为N的数组，却使用下标为N的位置
  - 因为某些错误，导致数组下标（变量）出错，从而越界
- 数组初值
  - 局部变量不会自动清零
  - 初始化设置的最大值太大，导致溢出
- 数组遍历
  - 访问的区间边界错误（终止条件：大于？大于等于？小于？小于等于？）

例子：YOJ-291

比利经常会碰到超大整数的加减法运算，而普通的计算器上无法进行。因此他想你帮他写一个程序来计算结果。

输入格式

输入共三行。 第一行为一个符号，表示要进行的计算（“+”表示要进行加法运算，“-”表示要进行减法运算）。 第二行第三行每行为一个大整数（长度小于2000），表示要进行加减操作的两个数。注意：整数既可以为正数也可以为负数。

输出格式

加减运算后的结果。

输入样例

+

99887766554433221 1

11223344556677

输出样例

998888888888888888

例题：YOJ-291

- 题意：
  - 高精度加法
- 思路（YOJ-291.c）：
  - 输入运算符和两个字符串
  - 判断两个数的符号
  - 将减法转化位加法（反转第二个操作数的符号）
  - 分类讨论每种情况应该要加还是减
  - 数组模拟计算
  - 输出答案

典型错误

- 没有考虑“\0”字符
- 定义：

```
const int N = 2000;

int len_a, len_b, len_ans;
char a[N], b[N], op, op_a, op_b;
char ans[N], op_ans;
```

- 使用：

```
scanf ("%s", a); len_a = strlen(a);
scanf ("%s", b); len_b = strlen(b);
```

典型错误

- 数组越界

```
void get_flag() {
    if (a[len_a] == '-') {op_a = '-'; len_a--;}
    else if (a[len_a] == '+') {op_a = '+'; len_a--;}
    else op_a = '+';

    if (b[len_b] == '-') {op_b = '-'; len_b--;}
    else if (b[len_b] == '+') {op_b = '+'; len_b--;}
    else op_b = '+';

    if (op == '-') {
        if (op_b == '-') op_b = '+';
        else op_b = '-';
    }
}
```

典型错误

- 由于某些错误，导致下标错误

```
void reverse() {
    for (int i = 0; i + i < len_a; ++i) {
        char t = a[i];
        a[i] = a[len_a - i];
        a[len_a - i] = t;
    }

    for (int i = 0; i + i < len_b; ++i) {
        char t = b[i];
        b[i] = b[len_b - i];
        b[len_b - i] = t;
    }
}
```

典型错误

- 访问区间错误

```
while (len_ans > 1 && ans[len_ans - 1] == '0') len_ans--;
```

```
if (op_ans == '-') putchar('-');
for (int i = strlen(ans) - 1; i >= 0; --i) {
    putchar(ans[i]);
}
putchar('\n');
```

调试练习-3

- YOJ-291：大整数加减（演示）
- YOJ-90：刷题高手（练习）
- YOJ-486-1：碱基串排序（练习）
- YOJ-486-2：碱基串排序（练习）

练习：YOJ-90

- 题意：
  - 老师给定n道题，一共有m位同学，第i位同学做了pi道题，老师想找到做题数（老师布置的题）最多的k位同学
- 思路（YOJ-90.c）：
  - 先读入老师的题，然后读入每位同学的题
  - 从同学的题中找出老师布置的题目，进行统计
  - 使用冒泡排序找出前k位同学

练习：YOJ-486

- 题意：
  - 给定A、G、C、T的优先级关系
  - 给定n个有A、G、C、T组成的字符串，对这些字符串进行排序
- 思路1（YOJ-486-1.c）：
  - 根据A、G、C、T的优先级关系，重构字符串
  - 根据新的字符串进行排序
- 思路2（YOJ-486-2.c）：
  - 枚举出所有情况，根据每种情况写不同的代码

4.5 函数、递归与分治调错

- 出错类型总结
- 例题：YOJ-123
- 练习：
  - YOJ-112
  - YOJ-119
  - YOJ-120
  - YOJ-123

出错类型总结

- 变量相关
  - 递归中使用修改相关变量，回溯时应该恢复
  - 局部变量和全局变量混淆使用
  - 局部变量未初始化
- 递归结束后没有使用return返回相关变量的值
  - 递归结束忘记返回
  - if-else分支遗漏某些情况，导致程序没有返回值
- 运行时错误（Segmentation fault）
  - 申请局部变量（数组），申请空间过大
  - 递归边界出错，无限递归

例题：YOJ-123

题目描述

一个古老的数学问题：

有三根杆子A，B，C。A杆上有N个(N>1)穿孔圆盘，盘的尺寸由下到上依次变小。要求按下列规则将所有圆盘移至C杆：
每次只能移动一个圆盘；
大盘不能叠在小盘上面。
提示：可将圆盘临时置于B杆，也可将从A杆移出的圆盘重新移回A杆，但都必须遵循上述两条规则。
问：如何移？最少要移动多少次？

输入格式

一行，包括一个整数N（N < 15）

输出格式

若干行，每行输出一次移动的步骤，输入格式如下：
[step %d] move plate %d# from %c to %c

最后一行输出一个整数，表示最少移动次数。

例题：YOJ-123

- 题意
  - 有n个圆盘，自底向上半径逐渐缩小；有3根柱子A、B、C。
  - 初始状态，这n个圆盘都在A号柱。需要借助B号柱将圆盘从A号柱移动到C号柱
- 思路
  - 使用函数 move(A, B, C, n, step) 表示将n个圆盘从A号柱经过B号柱移动到C号柱，已经移动了step步
  - 初始情况：move('A', 'B', 'C', n, 0)
  - 递归：A -> C -> B; n: A -> C; B -> A -> C
  - 递归边界：n==1

典型错误

- 递归边界出错（line 9）
- 变量重名（step变量）
- 没有返回值（line 14）

```
3   int step;
4
5   int move(char A, char B, char C, int n, int step) {
6       if (n == 1) {
7           step += 1;
8           printf ("[step %d]\t move plate %d#\t from %c to %c\n", step, n, A, C);
9       }
10      step = move(A, C, B, n - 1, step);
11      step += 1;
12      printf ("[step %d]\t move plate %d#\t from %c to %c\n", step, n, A, C);
13      int total = move(B, A, C, n - 1, step);
14  }
```

练习：YOJ-112

- 题意
  - 使用二分查找找到需要的数
- 思路（yoj-112.c）
  - 对于区间[left, right]，判断区间中点(left + right) / 2和需要查找的值之间的关系

练习：YOJ-119

- 题意
  - 使用递归的方式将字符串进行翻转
- 思路（yoj-119.c）
  - 当字符长度等于1时，直接返回
  - 否则，调换首尾两个字符，在递归地倒置字符数组的剩下部分

练习：YOJ-120

- 题意
  - 使用递归进行选择排序
- 思路
  - 递归边界：只有一个元素时结束递归
  - 对于区间[l, r]，选取其中最小的元素，和左端点的元素进行交换。对区间[l+1,r]继续递归
  - 答案在回溯的过程中输出

4.6 搜索算法的调错

- 出错类型总结
- 例题：YOJ-832
- 练习：
  - YOJ-113
  - YOJ-122
  - YOJ-117
  - YOJ-268
  - YOJ-437

出错类型总结

- 递归
  - 递归中使用修改相关变量，回溯时应该恢复
  - 结束条件成立时，没有return
  - 递归边界出错
- 队列
  - 起始时没有压入起始节点
  - 枚举遍历当前节点的孩子节点后，未弹出当前节点
- 剪枝
  - 限制搜索深度，避免超时
  - 及时止损，超出当前得到的最优解直接返回
  - 引导搜索方向，减少搜索范围

练习：YOJ-113

- 题意：
  - 在 n*n 的棋盘上摆放 n 个皇后的方案数
- 思路：
  - 按行搜索，遍历可摆放的列
  - 使用两个一维数组记录对角线是否安全

练习：YOJ-122

- 题意：
  - 由0和1组成的n*m的矩阵，表示迷宫，其中0表示墙壁（不通），1表示道路（坐标（1，1）和坐标（n，m）处都为1）
  - 若该迷宫存在从坐标（1，1）到坐标（n，m）的由数字1组成的通路，则输出YES，否则输出NO
- 思路：
  - 使用方向数组 `dir[4][2] = {{0,1},{1,0},{0,-1},{-1,0}}`
    - `x + dir[i][0]`
    - `y + dir[i][1]`
  - 尝试使用广搜，或许更方便

练习：YOJ-117

- 题意：
  - 小猴子现在想要从最低层开始一直爬到树顶（也就是最上面的那个枝条），摘尽可能多的桃子
- 思路：
  - 每次递归有两个分支，复杂度 2^n
  - 内部树枝节点会被访问多次，也就是不同递归分支来到了一个相同状态的递归节点
  - 剪枝：使用二维数组记录搜索到某个树枝的累计最大值，如果其他递归分支到了这个树枝，但是累计值小于等于记录的最大值，则停止递归

练习：YOJ-268

- 题意：
  - 图上搜索最短路径
- 思路：
  - 记录节点是否走过
  - 记录走过的路径

练习：YOJ-437

- 题意：
  - 设计一个最佳路线，能够让他取回一卡通，并且尽可能快地赶到上课地点(也就是走最短的路）
- 思路（YOJ-90.c）：
  - 两次搜索

例题：YOJ-832

- 思路
  - 使用深搜或广搜都有可能出现搜索范围过大导致超时
  - 考虑深搜并限制搜索深度
    - 针对某些测试点，搜索深度可能需要设置很大
  - 考虑逐步增加搜索深度
    - 但对于某些测试点，依旧会出现搜索范围过大
  - 考虑引导搜索方向
    - Tips：尽可能在当前已构造的纸带上搜索