我会自己 debug

随着程序设计题目越来越复杂，我们助教很有可能也无法在短时间内解决同学们的所有问题。例如为什么我输出的和答案不一样？为什么我的程序在中途异常终止了？等等。等到程序设计真正进入比较困难的阶段时，很有可能编码的时间只占据40%，而查错的时间要占据60%。因此，会自己查找解决错误是一项很重要的技能。接下来我将给大家介绍一些自主查错的技巧

一、构造数据，找出错误

为什么我本地样例测试都对，但是提交的时候总是`WA`呢？

这是因为样例是很有局限性的，没有涵盖题目涉及的所有情况。所以不要再说什么“本地样例是对的”了。

最典型的是《C2-G：解方程2023》，这一道题分类情况很多，需要同学们细心讨论。有的同学在写代码时没有考虑到$a=b=0$的情况，有的同学没有考虑到$a<0$的情况，等等。这时就需要我们自己给自己构造出一些测试数据，来观察自己的程序运行结果是否和自己预料的一致，如果不一致，那么这组数据极有可能反映出你程序的某种错误。

可以说，如果把debug的过程比作在海边捡拾贝壳，那么构造出一组错误的数据，就像你在看到了一枚贝壳，而剩下需要做的只是弯下腰去把贝壳捡起来而已。

接下来，我们将以下面的题目为例，给大家介绍debug的具体步骤。

```
# 大小写转换

## 题目描述
给定一个长度未知的字符串 A（含空格），请你对它进行大小写转换操作后得到字符串 B 并输出。大小写转换操作定义如下：
对 A 的每个字符：
 - 若该字符为小写字母，将其转换为大写字母。
 - 若该字符为大写字母，将其转换为小写字母。
 - 若该字符既不为小写字母，又不为大写字母，则该字符保持不变。

## 输入
一行，长度若干的字符串 A （含空格）。

## 输出
一行，字符串 A  经大小写转换操作后得到的字符串 B

## 输入样例1
To be or not to be, that's a question.

## 输出样例1
tO BE OR NOT TO BE, THAT'S A QUESTION.
```

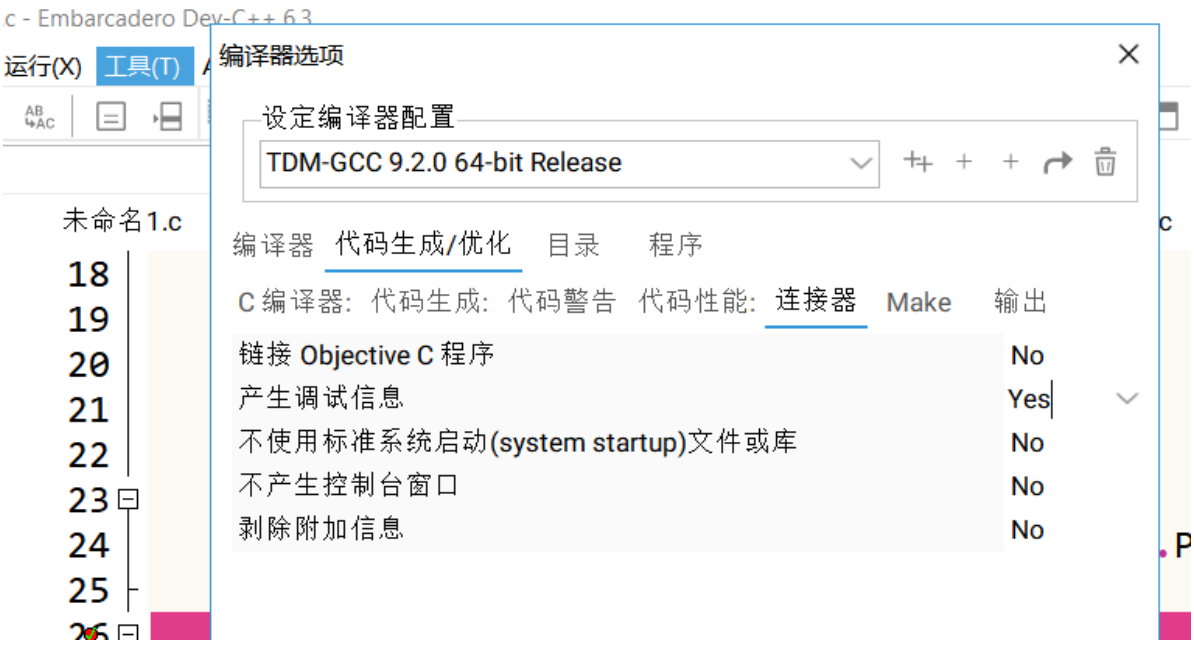
```
## 输入样例2
abCDeFg1234567

## 输出样例2
ABcdEfG1234567
```

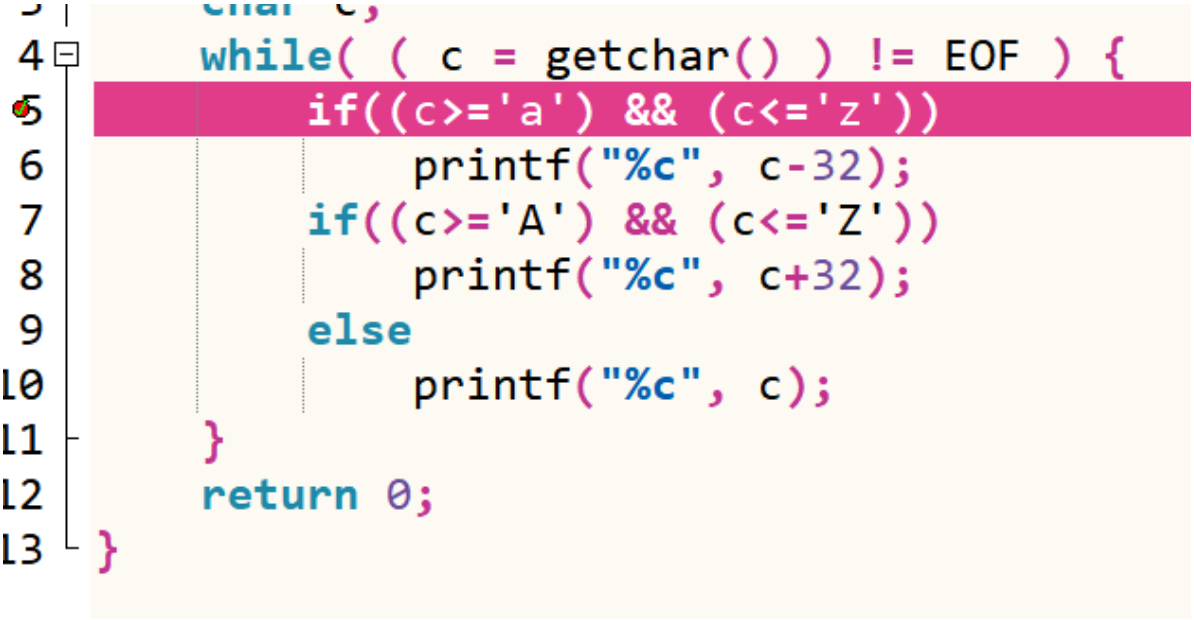
二、单步运行，观察过程

主流的IDE几乎都有单步运行调试的功能，接下来我以`dev-cpp`为例给大家讲解一下怎么用。

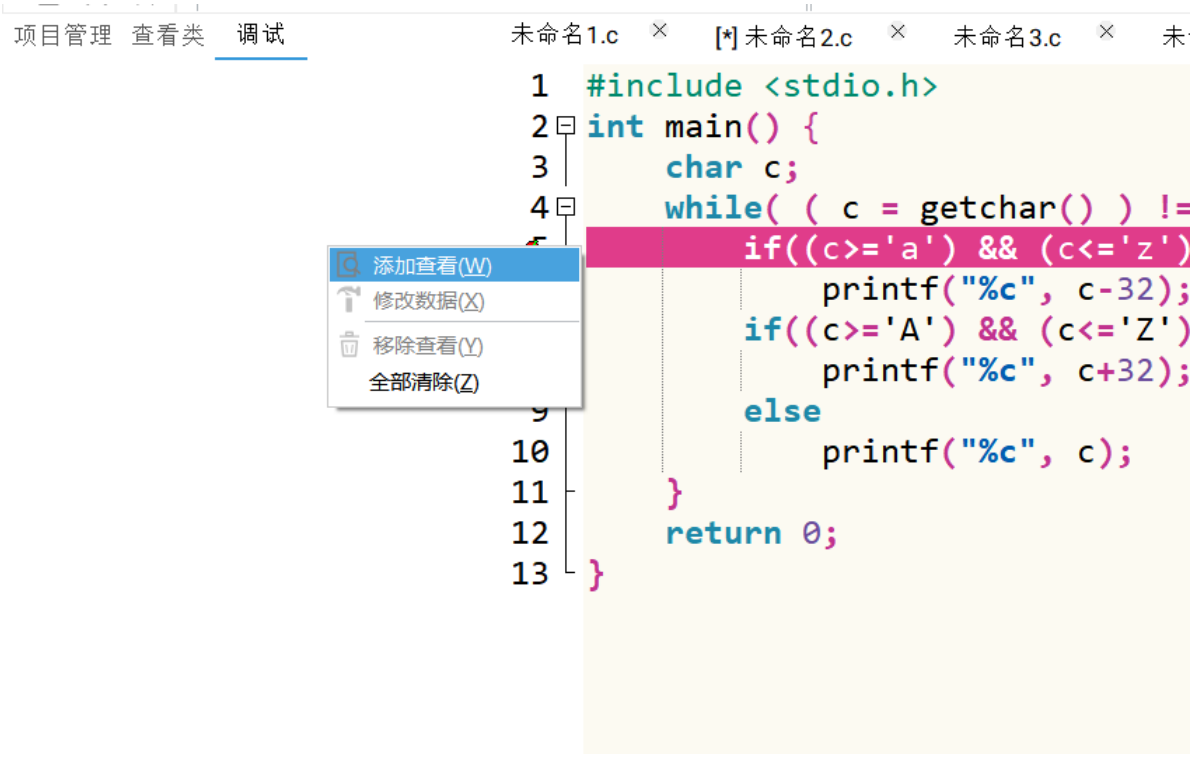
1. 在 工具-编译器选项-代码生成/优化-连接器 中打开“产生调试信息”开关。



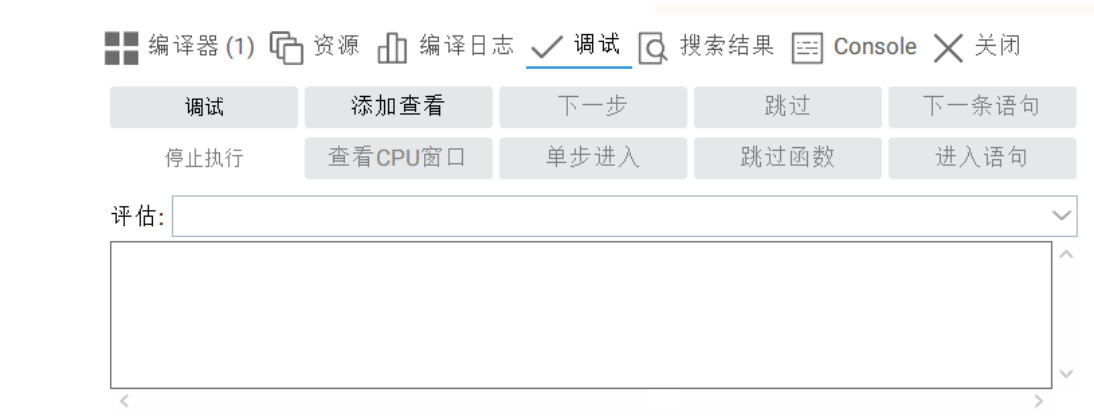
2. 单击你希望中断的行号，以此“设置断点”。



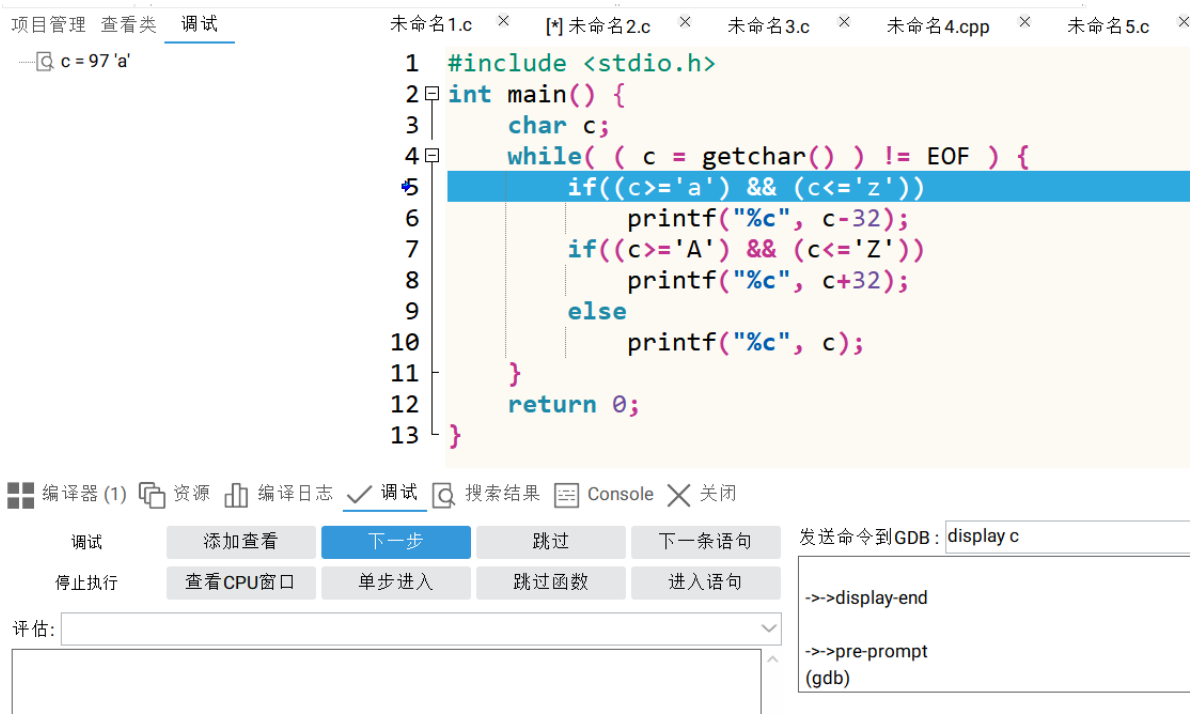
3. 在左侧“调试”处选择“添加查看”，来查看一个变量在程序运行过程中的值。



4. 点击下方的调试-调试，运行程序。



5. 点击“下一步”，单步执行，观察变量的变化过程和程序的执行情况。



三、输出调试，理清逻辑

由于咱们使用的IDE可能有些老旧，不太稳定，有时在调试时会出现异常的崩溃情况，这时候我们可以采取另一种更“简单粗暴”的方法，即输出调试法。

例如对《大小写转换》这题而言，示例的错误代码如下：

```
#include <stdio.h>
int main() {
    char c;
    while( ( c = getchar() ) != EOF ) {
        if((c>='a') && (c<='z')) {
            printf("%c", c-32);
        }

        if((c>='A') && (c<='Z')) {
            printf("%c", c+32);
        } else {
            printf("%c", c);
        }
    }
    return 0;
}
```

输入`a`，输出`Aa`.

为了找到问题，我们可以在每个`if`里面加一个输出标签，来确定程序进入了哪个`if`语句块中。

```
#include <stdio.h>
int main() {
    char c;
    while( ( c = getchar() ) != EOF ) {
        if((c>='a') && (c<='z')) {
            printf("1:");
            printf("%c", c-32);
        }

        if((c>='A') && (c<='Z')) {
            printf("2:");
            printf("%c", c+32);
        } else {
            printf("3:");
            printf("%c", c);
        }

    }
    return 0;
}
```

输入`a`，输出`1:A3:a`

这样一来，我们就知道程序进入了第一个if和最后一个else，可以据此来修改代码的逻辑。

当然，你也可以把关键变量的值在运行过程中就`print`出来，来观察这个变量的变化。

例如:

```
//......
double a,b;
//......
if(a/b<0.00001)
    printf("ERROR!\n");
else
    printf("%.4lf",a/b);
```

在输入一组数据时，程序输出`-nan`。

此时，我们可以在第4行前输出a和b的值，发现$a=b=0$，那么a/b的运算自然不能正常运行了，于是我们就精准定位了错误。

四、模块测试，稳扎稳打

有些复杂的题目，很可能由多个模块构成。我们可以在写完一个模块时，就构造一些数据，专门用来测试这个模块的运行情况。

这样debug，比整个程序都写完以后再慢慢定位问题，效率高很多。例如我要写一个四则运算计算器，那么写完加法部分后，就可以先测试测试加法的工作是否正常；当要输入字符时，如果你对不太有信心，可以输入以后立刻将被输入的东西输出，以此来查看是否正确进行了输入。

五、保持码风，清晰易懂

有些 IDE 有自动格式化代码的功能，例如 dev-cpp 可以按 `ctrl+shift+A` 来自动格式化代码。希望大家以后编程时都照着格式化后的这种格式来写，这样的话在编码时你的逻辑就很清晰了，不会自己把自己绕进去。

你觉得是看这种代码清楚呢?

```
#include <stdio.h>
int main() {
char c;
while( ( c = getchar() ) != EOF ) {
if((c>='a') && (c<='z'))
 printf("%c", c-32);

if((c>='A') && (c<='Z')) {
printf("%c", c+32);
                    } else
printf("%c", c);



    }
    return 0;


}
```

还是这种清楚呢?

```c
#include <stdio.h>
int main() {
    char c;
    while( ( c = getchar() ) != EOF ) {
        if((c>='a') && (c<='z'))
            printf("%c", c-32);
        if((c>='A') && (c<='Z')) {
            printf("%c", c+32);
        } else
            printf("%c", c);
    }
    return 0;
}
```

我相信大家都有自己的判断。

此外，还有几个小建议：

1. **写 `for,while,if,else` 这种语句时，无论后面写几行，都要加大括号。**

   有的同学可能会说，如果我只写一行，不是不用加吗？其实按理来说确实不用加，但是有的人就会犯这种错误：

   ```c
   #include <stdio.h>
   int main()
   {
       int i, j, a, b;
       scanf("%d %d",&i, &j);
       a = i/j + 1;
       b = i/j;
       if (i%j != 0);
           printf("%d", a);
       else
           printf("%d", b);
       return 0;
   }
   ```

   看到了吗？第8行的`if`后面多了一个分号。

   如果你一直加大括号，那么`if`后面永远是`{`。一旦你看到`if`后面没有`{`,你就能提示自己，这里可能出错了。

2. **对于复杂的运算式，搞不清楚运算符优先级？加括号！**

   这样可以保证运算的按照你想的顺序运行。凡是你自己不清楚的，那就加括号！

   尤其是位运算的优先级问题，需要特别的注意。

3. **尽可能把表达式写简单，清晰。把一个很长的表达式拆分成多个简单的表达式。**

   别写`a=(++i)+(++i)+(++i)+(i++)`了，求求你了，何苦自己为难自己呢？

4. **让编译器显示全部警告信息**

   真的，可有用了，绝大多数低级错误都能发现。当然在使用`%lld`输入`long long int`型时会误报，忽略就行。

六、寻求帮助，学会提问

前面写了这么多，并不是说就不让你提问了。如果真的自己也解决不了，百度也百度不到，一定要积极提问。但是在**提问时**，请至少**包含以下内容**：

1. **题号**

   比赛名称、题目编号或题目链接

2. **代码**

   别拍照，别截图

   （因为图片经常存在**不够清晰**的问题，不容易看到一些可能出问题的关键细节，另外图片中的代码在助教本地复现时需要耗费更多的时间，比较**影响答疑效率**）

   推荐**直接复制粘贴**，使用https://pastebin.ubuntu.com/

3. **报错信息**

   编译失败的报错信息、提交到OJ上没有AC的报错信息、代码评测得分，或者自己测试时遇到的错误等等，就是你遇到了什么问题

还可以包括以下内容，可以让我们更快速的定位问题：

4. **编程思路**
5. 你自己debug的过程和成果

不推荐的提问方式：

- 助教请问我这个代码为什么 CE 啊?
- 助教我本地样例是对的，但是提交到 OJ 上是 WA 。

推荐的提问方式：

- 助教我的代码编译错误，编译器说`[错误] 'a' undeclared (first use in this function)`，请问为什么啊?
- 助教我的代码如果输入`a`，本来应该输出`A`，但是它实际上输出了`Aa`，我找不出来问题在哪，你能帮帮我吗?

推荐阅读：提问的智慧