《C 程序设计》文件集合

（调试测试专题）

1. **DevC++的调试**
2. **GDB调试实例**
3. **DevC++单步调试**
4. **如何使用 freopen 生成数据和测试**
5. **如何用 printf 进行日志式调试**
6. **关于 ACM 的输入输出（2 则）**
7. **如何设计测试数据**

厦门大学信息学院软件工程系　黄炜　副教授

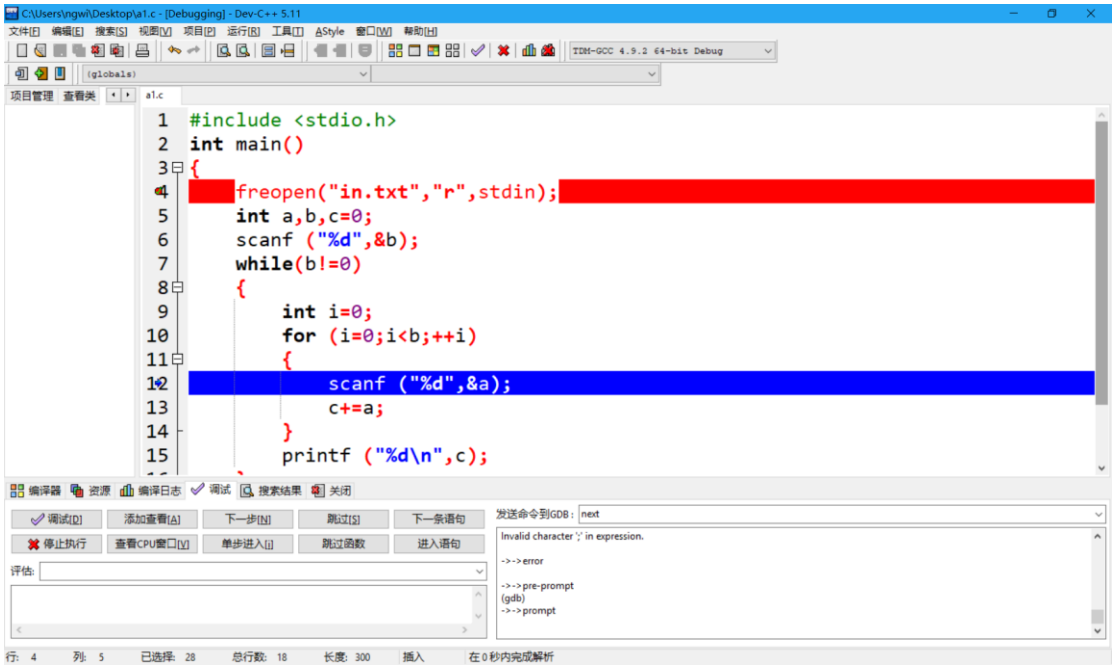
二零一九年十二月二十七日

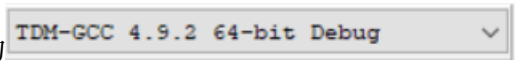
DevC++的调试

对 OJ 程序走向有疑问的同学可以使用单步调试和断点调试功能，了解程序的走向。

方法如下：

第一步，输入你的程序，在你程序的声明语句之后（如果 C99 允许在 main 函数的第一行），加上“freopen("in.txt","r",stdin);”，并在对应 in.txt 中输入你的测试输入，这样你就不必和输入搅在一起，更专心地调试程序了。

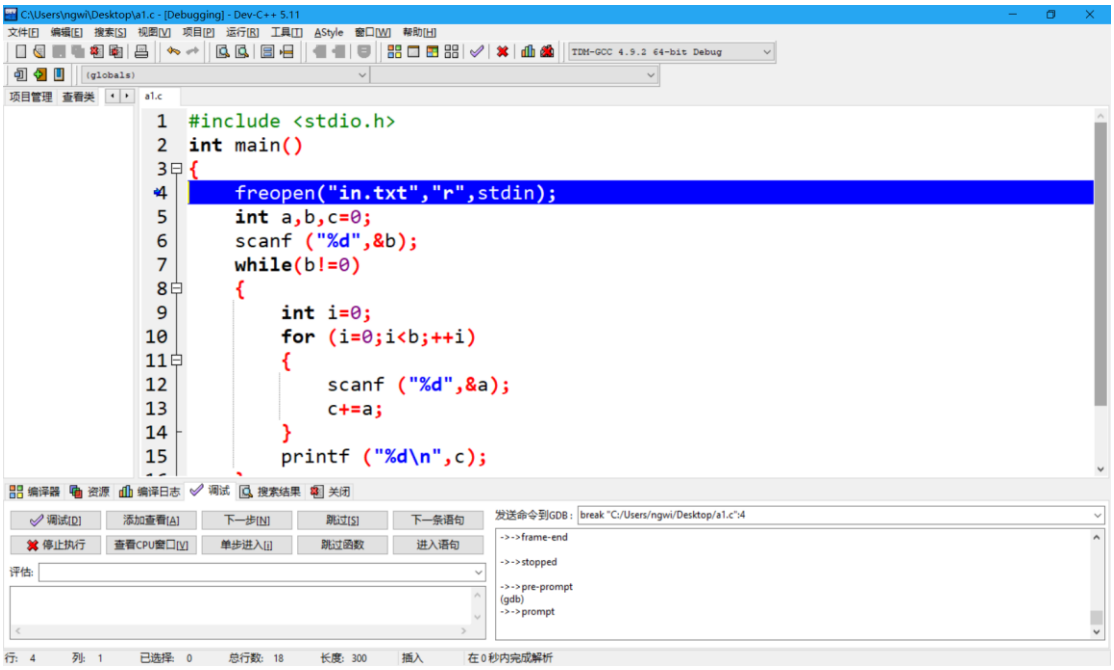


第二步，设置断点，在 freopen 这一行按下鼠标左键或者按下 F4 设置断点。同时，右上角的编译选项应改为 

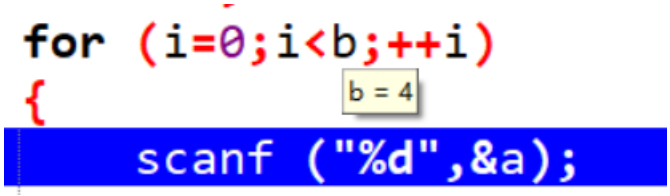
。注意：32bit 也可以，但一定要是 Debug，不是 Release 或者其它。

第三步，编译并以调试模式运行程序。先按下 F9 编译程序，再按 F5 调试程序。

第四步，实施调试。蓝色语句是断点所在位置，也就是程序运行到这一行之上一行。单击左下角工具箱中的按钮，可以分别实施操作。例如单击“下一步”按钮，可以看下一步程序执行的是哪一条语句。



如果你要查看变量的值，可以将鼠标停靠在其上方查看。



或者发送命令到 GDB。例如输入：p i 并回车（意思为 print i 的值）右下角窗口第一行即出现 0（意思是 i 为 0）。

GDB 调试实例

以下是利用 DevC++自带编译器的 GDB.EXE 命令行调试一个程序。Linux 系统通常可以安装 C 调试器 gdb。其内容应该是很接近的。

以下，蓝色部分是用户输入，黑色部分是系统输出。

```
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation。保留所有权利。

C:\Users\ngwi>set path=%path%;E:\Desktop\Dev-Cpp\Dev-Cpp

C:\Users\ngwi>E:

E:\>cd Desktop

E:\Desktop>gcc -g x1.c -o a.exe

E:\Desktop>gdb a.exe
GNU gdb (GDB) 7.8.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.o
This is free software: you are free to change and redist
There is NO WARRANTY, to the extent permitted by law.  T
and "show warranty" for details.
This GDB was configured as "x86_64-w64-mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources on
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "w
Reading symbols from a.exe...done.
(gdb) l
1       #include <stdio.h>
2       int main()
3       {
4               int avg,a,b,c;
5
6               scanf("%d",&a);
7               scanf("%d",&b);
8               scanf("%d",&c);
```

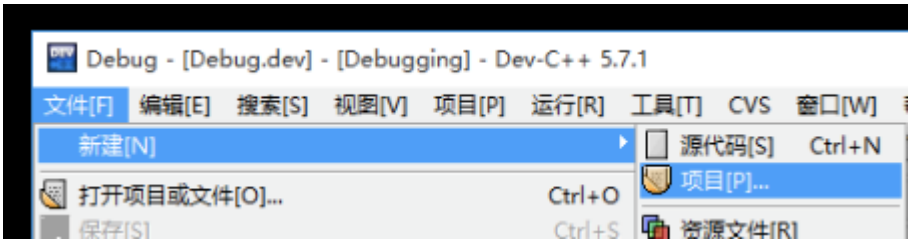
```
9
10              avg=(a+b+c)/3; // calculate the average
(gdb) b 6
Breakpoint 1 at 0x40153d: file x1.c, line 6.
(gdb) b 10
Breakpoint 2 at 0x401576: file x1.c, line 10.
(gdb) r
Starting program: E:\Desktop\a.exe
[New Thread 11324.0x236c]

Breakpoint 1, main () at x1.c:6
6               scanf("%d",&a);
(gdb) n

7               scanf("%d",&b);
(gdb) n

8               scanf("%d",&c);
(gdb) n

Breakpoint 2, main () at x1.c:10
10              avg=(a+b+c)/3; // calculate the average
(gdb) p a
$1 = 3
(gdb) p b
$2 = 0
(gdb) p c
$3 = 17
(gdb) n
12              printf("Average: %d\n",avg);
(gdb) p avg
$4 = 6
(gdb) p a+b+c
$5 = 20
(gdb) p 20/3
$6 = 6
(gdb) p 20/3.0
$7 = 6.666666666666667
(gdb) n
Average: 6
14              return 0;
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) n
```

```
Program not restarted.
(gdb) q
A debugging session is active.

        Inferior 1 [process 11324] will be killed.

Quit anyway? (y or n) y

E:\Desktop>
```

DevC++ 单步调试

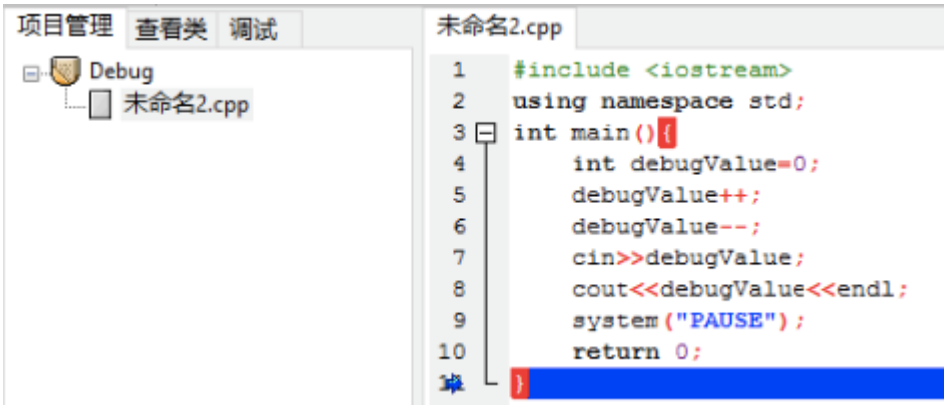
这个在较小的代码里相对有用,但是算法题和工程调试,还是建议输出调试－很多人都这么说.

- Step1 新建一个工程(可选)



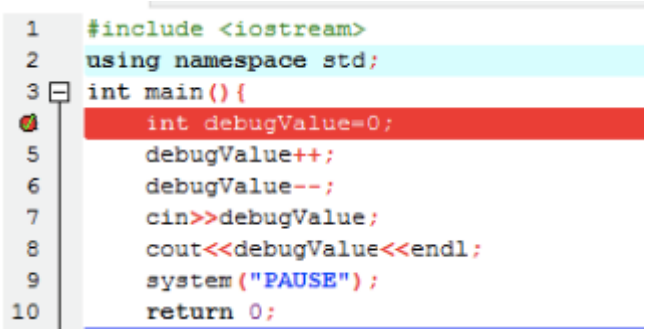
可能要调 dev 为支持 debug 模式

- Step2 敲入(粘贴..)一段调试代码



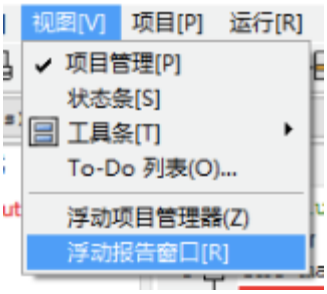

```
#include <iostream>
using namespace std;
int main(){
  int debugValue=0;
  debugValue++;
  debugValue--;
  cin>>debugValue;
  cout<<debugValue<<endl;
  system("PAUSE");
  return 0;
}
```

- Step3 点击某一行 Red it~



- Step4 在那一行选一个变量,右键”添加查看”(其实这里有很多其它方法)

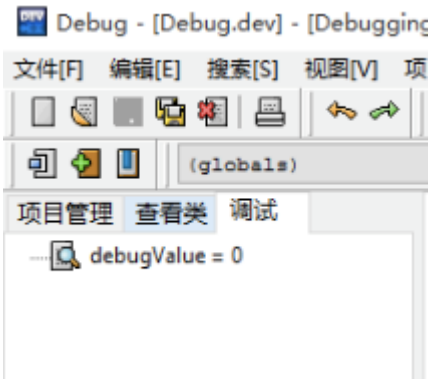
- Step5 把左边项目管理文件框切换到调试,把下面报告窗口(一般用来显示编译错误的地方)也切换到调试



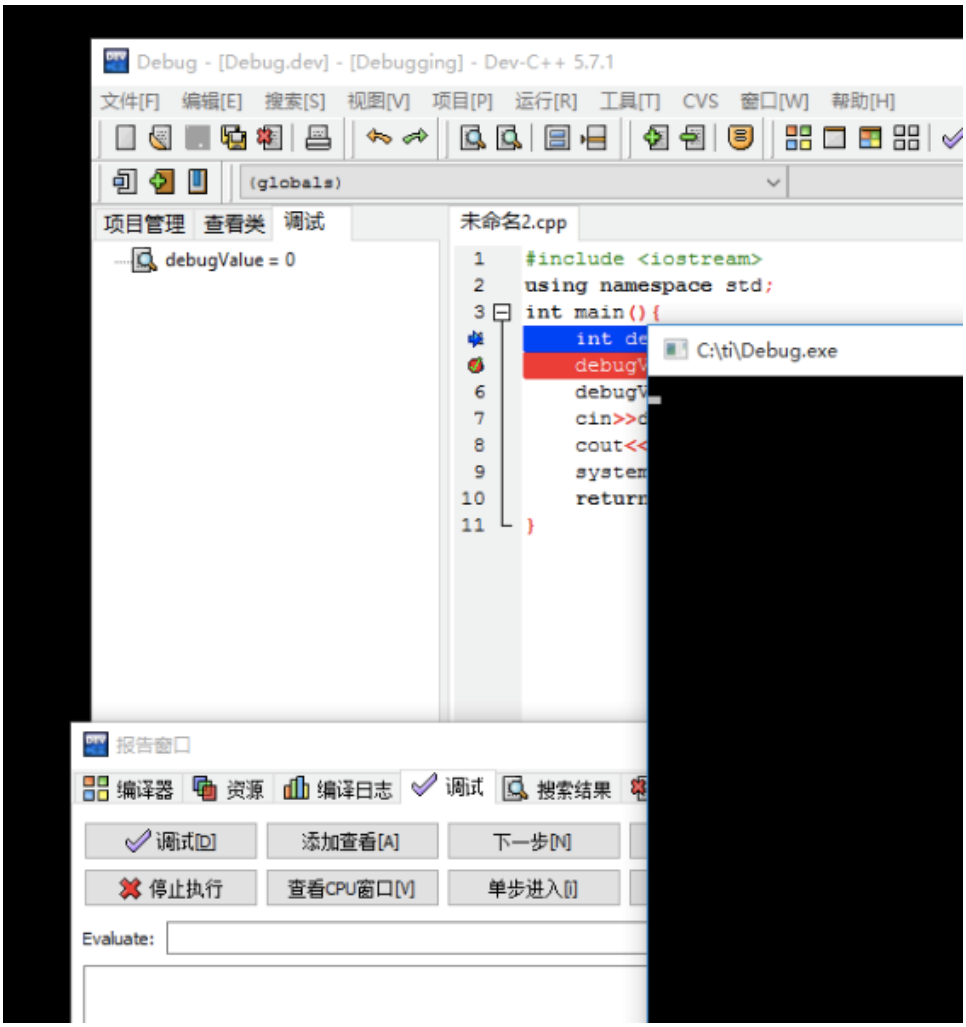
(如果没有那两个框的话在顶端视图栏打开)

- Step6 点击调试 弹出终端

- Step7 在下面的调试栏目点下一步观察左边栏目的调试区域变量值的变化



- Step8 当执行到 cin 时候,在弹出的命令行输入数字(如 123),可以发现值发生变化



- Fighting~

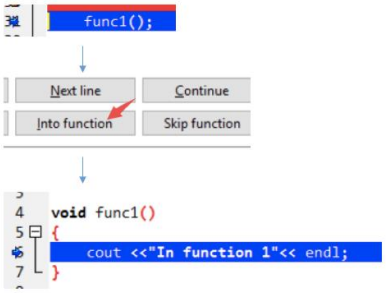
另外发现…写完这篇之后的下一周软设实验居然就讲这个…

补充一点其它技巧

三 按F7，执行下一句。或者点击
Next line

正在执行的语句为蓝色显示

四 遇到函数，可以点击into function
进入函数内部进行调试



如何使用 freopen 生成数据和测试

有同学询问 freopen 的使用方法，我在此对上课内容作总结回答。

OJ 做题是唯结果论的，主要分两步：

（1）猜测后台数据是什么；

（2）如何快速找到速度和时间在规定范围内的解题方法。

freopen 是将原本需要从键盘（stdin）导入或导出到屏幕（stdout）的文本，重定向到文件中，便于编辑和阅读。因此可以利用 freopen 生成测试输入文件，利用 freopen 将输入文件导入到程序中，利用 freopen 将输出导出到文件中，便于阅读比较。

我们以“寻找第二大的数”题为例，题意大致为寻找给定四个数中第二大的数（即小于最大值的数中最大的数），如果四个数相同则输出该数。

初步测试为样例输入的测试。由于样例输入字符较多，测试又需要反复进行，人工输入耗时费力。可以将测试文件下载下来（“样例输入”标题后有下载图标），在程序中使用“freopen("in.txt","r",stdin);”将输入导向文本文件。以后每次运行时，由文本文件向程序供应输入。

类似地，当程序输出较多，而输出到屏幕输出难以阅读比较的时候，可以使用“freopen("out.txt","w",stdout);”将输出导向文本文件。便于反复打开使用。

以该题为例，我们应当测试 0123、1023、1032、……等几种情况，但精确地划分测试类需要花费较多的思考时间，因此建议测试 0000~3333 共 4*4*4*4=256 种情形。做题人必须自制输入文件，同样利用 freopen，书写“freopen("in.txt","w",stdout);”并运行程序。即：

```
#include <stdio.h>
int main()
{
    freopen("in.txt","w",stdout);
    for (int i=0; i<4; i++)
        for (int i2=0; i2<4; i2++)
            for (int i3=0; i3<4; i3++)
                for (int i4=0; i4<4; i4++)
                    printf("%d %d %d %d\n",i,i2,i3,i4);
}
```

则生成了输入文件，下次运行时，可以作为输入文件使用。

```
#include <stdio.h>
int main()
{
    freopen("in.txt","r",stdin);
    freopen("out.txt","w",stdout);
    （程序主体）
}
```

运行后，逐行检视，最终确定有问题的输入。再将 in.txt 清空，将有问题的输入写入。用断点调试（含：单步调试）调试程序，确定程序在哪里出问题。

如何用 printf 进行日志式调试

以提交号 54346 为例，程序输入：

```
1 2 3 4
2 3 4 5
7 3 2 1
1 1 1 1
5 5 5 3
```

程序输出：

```
4
1
3
5
```

预期的输出是：

```
4
1
3
3
```

程序是：（记得先格式化）

```c
#include <stdio.h>

int main(void)
{
    int a, b, c, d, x, y, z, p;
    while (scanf("%d %d %d %d", &a, &b, &c, &d) != EOF)
    {
        if (a <= b)
        {
            x = a;
            a = b;
            b = x;
        }
        if (c <= d)
        {
            y = c;
            c = d;
            d = y;
        }
        if (a >= c)
        {
```

```
            z = a;
            a = c;
            c = z;
        }
        if (b <= d)
        {
            p = b;
            b = d;
            d = p;
        }
        if (a <= b)
            a = b;
        printf("%d\n", a);
    }
    return 0;
}
```

在重要位置先加上打印__LINE__和状态（a, b, c, d, x, y, z, p 的值）。

```
#include <stdio.h>

int main(void)
{
    int a, b, c, d, x, y, z, p;
    while (scanf("%d %d %d %d", &a, &b, &c, &d) != EOF)
    {
        printf("%d: %d %d %d %d %d %d %d %d\n", __LINE__, a, b, c, d, x, y, z, p);
        if (a <= b)
        {
            x = a;
            a = b;
            b = x;
            printf("%d: %d %d %d %d %d %d %d %d\n", __LINE__, a, b, c, d, x, y, z, p);
        }
        if (c <= d)
        {
            y = c;
            c = d;
            d = y;
            printf("%d: %d %d %d %d %d %d %d %d\n", __LINE__, a, b, c, d, x, y, z, p);
        }
        if (a >= c)
        {
            z = a;
            a = c;
            c = z;
            printf("%d: %d %d %d %d %d %d %d %d\n", __LINE__, a, b, c, d, x, y, z, p);
        }
        if (b <= d)
```

```
        {
            p = b;
            b = d;
            d = p;
            printf("%d: %d %d %d %d %d %d %d %d\n", __LINE__, a, b, c, d, x, y, z, p);
        }
        if (a <= b)
        {
            printf("%d: %d %d %d %d %d %d %d %d\n", __LINE__, a, b, c, d, x, y, z, p);
            a = b;
        }
        printf("%d\n", a);
        printf("%d: %d %d %d %d %d %d %d %d\n", __LINE__, a, b, c, d, x, y, z, p);
    }
    return 0;
}
```

输入：

```
5 5 5 3
```

输出

```
8: 5 5 5 3 1 3 0 1
14: 5 5 5 3 5 3 0 1
28: 5 5 5 3 5 3 5 1
39: 5 5 5 3 5 3 5 1
5
```

即先运行第 8 行，然后第 14、28、39 行。

说明：(a <= b) 、(a >= c)、(a <= b) ，但这样的第 2 个数（b）不是第二大的数。这是整个程序的问题，应再补强。

关于 ACM 的输入输出（2 则）

1 关于 ACM 的输入输出（一）

写给第一次参加现场赛的同学们

一般来说 ACM 的现场赛会规定输入输出

或者是文件输入标准输出

也可能是文件输入文件输出

如果没有规定的话那么一般就是标准的输入输出了

那说一下输入输出的重定向

一般用下面两种方法

c++常用:

```
#include <fstream.h>
ifstream filein("data.in");  // 定义一个文件输入流
ofstream fileout("data.out"); //cout<< --> fileout<<
filein.eof() //文件到末尾,返回非零值
```

data.in 表示输入的数据文件

本地测试的话本来输入的数据就要在这个文件里面测试了

建一个本地的文本 data.in,可以用记事本的方式打开

注意:文件输入的话,以后的 cin>>都要改成 filein>>, cout<<都要改成 fileout<<

c 语言常用:

```
freopen("date.in","r",stdin);  //重定向所有标准的输入为文件输入
freopen("date.out","w",stdout);//重定向所有标准的输出为文件输出
fclose(stdout);//输出结束
```

```
freopen("date.in","r",stdin); //重定向所有标准的输入为文件输入
freopen("date.out","w",stdout);//重定向所有标准的输出为文件输出
fclose(stdout);//输出结束
```

第一句的意思就是文件输入,以"读状态",去替换标准的输入

以上如果只是规定用文件输入输出 的某一种,那么就只用其中的一种

2 关于 ACM 的输入输出（二）

ACM 题目特点: 由于 ACM 竞赛题目的输入数据和输出数据一般有多组（不定），并且格式多种多样，所以，如何处理题目的输入输出是对大家的一项最基本的要求。这也是困扰初学者的一大问题。 ACM 的输入输出要求严格按照规定来，所以你不需要输出像"Please input the data"这类的提示语。否则将会被判 Wrong Answer。

1、输入

初学者一般有个误区：如果题目包含多组测试数据，他们就会把输入的内容全部保存起来，然后再依次处理。

其实程序的输入/输出是相互独立的，因此，每当处理完一组测试数据，就应当按题目要求进行相应的输出操作。而不必将所有结果储存起来一起输出。

下面来介绍一下 ACM 中常见的一些输入情况。

只有一组测试数据

这类题目是最简单的，比如第 1000 题。参考代码：

```
#include
int main(void)
{
```

```
int a, b;
scanf("%d %d", &a, &b);
printf("%d/n", a + b);
return 0;
}
```

没有明确指出输入什么时候结束

如果是这种情况，我们默认是以“文件结束”(EOF)为结束标志。

这是 ACM 的默规，例如 1076 题。参考代码：

```
#include
int main(void)
{
int a, b;
while (scanf("%d %d", &a, &b) != EOF)
printf("%d/n", a + b);
return 0;
}
```

指定数据量

有时会在数据的第一行提供数据量大小，比如第一行是 100，则表示有 100 组数据。比如第 1077 题。参考代码：

```
#include
int main(void)
{
int n, a, b;
scanf("%d", &n);
```

```
while (n--)
{
scanf("%d %d", &a, &b);
printf("%d/n", a + b);
}
return 0;
}
```

以特定元素作结束符

这种输入和第一种类似。常见的是规定以 0 作为结束符。

比如第 1078 题。参考代码：

```
#include
int main(void)
{
int a, b;

while (scanf("%d %d", &a, &b), a || b)
printf("%d/n", a + b);

return 0;
}
```

输出

输出格式统一

这种比较简单，只要按要求来就没问题的。

比如每组输出占一行，或者每组输出后面加一个空行。比如 1000 题。

数据之间有空行

对于这种输出，有时候还会告诉你有几组输入，这样你就可以自己判断一下是不是最后一组。是就不输出空行，否则多输出一个空行。而有时候连共有几组数据都不会告诉你。其实不论知不知道有几组数据，我们都可以这样处理。

第一组数据后面不加空行。

第二组开始，每组前面加空行。 比如第 1079 题，参考代码：

```
#include
int main(void)
{
int a, b, i = 0;
while (scanf("%d %d", &a, &b), a || b)
printf((i++? "/n%d/n": "%d/n"), a + b);
return 0;
}
```

3 关于 ACM 的输入输出（三）

在线判决系统是机器判题系统，也就是俗称的 OJ（Online Judge），机器判决的一个特点就是必须 100%的吻合才能判为正确，否则要么 WA,PE。同时对于提交的程序还有一定的时间限制，如果超过时间则会判超时。OJ 一般采用的是标准输入输出，所以提交的

时候我们不必要使用文件读入输出（这与高中的信息学是不同的），机器判决只针对程序结果，不针对程序，所以很多时候直接提交数据也是可以的，俗称打表。

下面介绍常用的处理输入的方法

几种常用的处理输入方法(C 语言)

感觉新人对于处理输入输出存在一些问题,这里写出几个常用到的处理方法:

1.知道输入数据组数 n

```
scanf("%d",&n);
whlie(n--){
   这里处理每一组输入.然后直接按格式输出,没必要开数组存储答案.
}
```

2.没有数据总数,以 EOF 结束

可能用的几个函数:

scanf():

```
while(scanf("%s|%d")!=EOF){
  处理每一组数据,并输出.
}
```

getchar():读入一个字符

```
whlie((ch=getchar())!=EOF){
```

}

gets():读入一行

while(gets(buf)!=NULL) {

}

用 getchar,gets 注意读入换行符.

3.以 0 或-1 结束的输入.

while(scanf("%d",&n),n!=0) {

}

关于 C++的输入输出处理：

cin 读字符串时遇到空白符（空格，换行等）结束

```
char str[BUFFER];
while (cin >> str) {
}
```

getline 读字符串时遇到换行符结束,用于读一整行

```
char str[BUFFER];
while (cin.getline(str, BUFFER)) {
}
string str;
while (getline(cin, str)) {
}
```

cin/cout 要比 scanf/printf 慢一些，尽可能使用 scanf/printf 以避免测试大量数据时因为输入输出慢而导致 TLE. putchar/getchar 要比 scanf/printf 更快

关于 java 的输入输出处理：

如果使用 BufferedReader(jdk1.1 或以后的版本，一次读一整行字符串，类似于 gets)

```
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
String s;
while ((s = stdin.readLine()) != null) {
   可以用 StringTokenizer st = new StringTokenizer(s);来按空格切词
   int n = Integer.parseInt(st.nextToken());
   double b = Double.parseDouble(st.nextToken());
}
```

如果使用 Scanner(仅限于 jdk1.5 或以后的版本，一般用于从字符串中切词，类似于 cin)

```
Scanner stdin = new Scanner(System.in);
while (stdin.hasNext()) {
   String s = stdin.next();
   int n = stdin.nextInt();
   double b = stdin.nextDouble();
}
```

至于输出，很多新手总会选择先将答案存储在一个数组里，等程序运行完再输出，其实这是没有必要的，机器判决是逐个字符匹配，所以完全可以处理一组输入后，便输出结果。

ACM 技巧 使用文件输入输出方便测试的方法

把下面两块宏语句分别嵌在 main 函数的开始和结束，这样在本地调试的时候，cin/cout 和 scanf/printf 直接对应到指定的文件流，但提交到 OJ 时，此两句不被编译，所以仍为标准 I/O 流，因此不用提交前改代码。

后面一块宏不用也可以，前面一块宏根据自己的输入文件改变"in.txt","out.txt"，也可以只用其一。

```
#include <iostream>
#include <cstdio>

using namespace std;

#ifndef ONLINE_JUDGE
    freopen("in.txt","r",stdin);
    freopen("out.txt","w",stdout);
#endif

#ifndef ONLINE_JUDGE
    fclose(stdin);
    fclose(stdout);
```

#endif

用这种方法，cin/cout 和 scanf/printf 都可以转化为文件流

C 语言输入输出函数详解

C 语言中基本的输入输出函数有：

putchar ():把变量中的一个字符常量输出到显示器屏幕上;

getchar ();从键盘上输入一个字符常量,此常量就是该函数的值;

printf ();把键盘中的各类数据,加以格式控制输出到显示器屏幕上;

scanf ();从键盘上输入各类数据,并存放到程序变量中;

puts ():把数组变量中的一个字符串常量输出到显示器屏幕上;

gets ():从键盘上输入一个字符串常量并放到程序的数组中.

sscanf(); 从一个字符串中提取各类数据。

putchar() 和 getchar() 顾名思议就是从输入流中获取一个字符和输出一个字符，比较简单，不再多讲。

例子如下：

```
char c = getchar();
putchar(c);
```

格式化输入输出 scanf()和 printf()是最有用的，所以重点讲一下。

printf():

一般形式:

printf("格式控制".输出列表);

eg : printf("a=%d,b=%f,c=%c/n",a,b,c);

1;格式控制.

格式控制是用双引号括起来的字符串,也称"转换控制字符串",它包含以下两部分信息.

格式说明：由"%"和格式字符组成,如%d,%f,%c,他的作用是把输出数据转换为指定格式输出,格式的说明总是由"%"字符开始的.

普通字符：需要原样输出的字符,或者是一些有特殊含义的字符，如/n,/t。

2;输出列表

就是需要输出的一些数据,也可以是表达式,如果在函数中需要输出多个变量或表达式,则要用逗号隔开.

一些特殊字符的输出：

单引号，双引号，和反斜杠的输出在前面加转义字符”/”

如：”/”’，“/””，“//”

%的输出用两个连在一起的%%，即 printf(“%%”);

常用的格式说明如下:

格式字符

d 以十进制形式输出带符号整数(正数不输出符号)

o 以八进制形式输出无符号整数(不输出前缀 O)

x 以十六进制形式输出无符号整数(不输出前缀 OX)

u 以十进制形式输出无符号整数

f 以小数形式输出单精度实数

lf 以小数形式输出双精度实数

e 以指数形式输出单、双精度实数

g 以%f%e 中较短的输出宽度输出单、双精度实数

c 输出单个字符

s 输出字符串

这里强调一下：网上很多文章都说 f 和 lf 是一样的，即不管单精度，双精度浮点数，都可以用 f, 但我在 POJ 上做过测试，输出 Double 时用 f 确实也可以 ，但读入时，用 f 就报 WA，所以大家如果对 Double 进行读写的话，都用 lf 吧。说到 Double，再啰嗦一句，建议大家要用到浮点数时都用 Double，不要用 float，因为在很多情况下，float 精度不够会导致 WA。

特殊：

对 64 位整数的输入输出，在 POJ 上的 C++环境下(即 VC)，64 位整数是：

__int64 （注意 int 前面是两个下划线）

输入输出格式为"%I64d".

在 G++环境下(即 Dev C++) 64 位整数是

long long

输入输出格式为"%lld".

输出宽度

用十进制整数来表示输出的最少位数。 注意若实际位数多于定义的宽度，则按实际位数输出， 若实际位数少于定义的宽度则补以空格或 0。

精度

精度格式符以"."开头，后跟十进制整数。意义是：如果输出数字，则表示小数的位数；如果输出的是字符， 则表示输出字符的个数；若实际位数大于所定义的精度数，则截去超过的部分。

标志格式字符

- 结果左对齐，右边填空格

+ 输出符号(正号或负号)空格输出值为正时冠以空格，为负时冠以负号

例如：

double c=24212345.24232;

printf(“%020.4”); 表示输出精确到小数点后 4 位，输出占 20 位，若有空余的位补 0.

scanf：

scanf 的很多用法都是和 printf 对应的，故不再赘述。

说一下 scanf 一个特别好用的地方，就是可以滤去一些不想要的东西。

举例说明如下：

比如输入为日期 yyyy-mm-dd，就可以这样写：

int year,moth,day;

scanf(“%d-%d-%d”,&year,&moth,&day);

再比如：

scanf("%3d %*3d %2d",&m,&n);　输入 113 118 69 回车(系统将 113 赋予 m,将 69 赋予 n,因为*号表示跳过它相应的数据所以 118 不赋予任何变量)

puts()用的不多，且基本都能用 printf()代替，故不再多说。

gets()是从输入流中获取一行字符串放入字符数组中:

char in[100];

gets(in);

大家可能最容易出错的地方就是字符串的输入，所以强调一下：

能进行字符，字符串输入的有：

getchar(), scanf(“%c”); scanf(“%s”), gets()

其中 getchar() 和 scanf(“%c”)的功能是一样的。

需要注意的是，这两个函数读入的是输入流中当前位置的字符，

比如：

scanf(“%d”,&n);

c = getchar();

假设输入 67/ (假设“/”代表回车)，则第一个 scanf 读入一个整数 67 后，当前输入流的位置是 67 之后，即指向回车符，所以第二个 getchar()读入的就是一个回车符了，即 c = ‘/n’。

同样，gets()也是从当前位置读入一行字符串。

比如：

scanf(“%d”,&n);

gets(str);

此时读入字符数组中的字符串就是“/n” 了

所以通常在用 scanf 读入一个非字符串的类型之后，如果要读入字符，或字符数组，都用一个额外的 getchar()把回车符读掉，若后面跟的不止一个回车符，可能还有多余的空格的话，就用 gets()读掉。

和以上不同的是，scanf(“%s”) 读入的时候是会忽略掉空格，回车和制表符的。并且以空格，回车和制表符作为字符串结束的标志。

经常会有这样的题，输入第一行是一个整数，接下来每行的第一个是一个字符，用来表示某种操作，后面再跟一些数据，比如：

4

A 100 2

B 23

A 23 89

B 34

像这种输入就需要小心，读入字符时不要读成回车符。

为了防止意外，我一般是这样处理这类输入的：

```
char model[2];
Scanf(“%d”,&n);
for(…,…,…){
   scanf(“%s”,model);
   if(model[0] == ‘A’){
}
else{
}
}
```

sscanf():

sscanf()经常用来分解字符串，功能非常强大，但很多功能都需要正则表达式的知识，所以就介绍一下最简单的几种用法，大家如果想了解更多的话，自己去网上找吧。

1、

```
char str[100],str1[100],str2[100];
gets(str);
sscanf(str,”%s%s”,str1,str2);
```

将读入的一整行字符串按空格，制表符或回车符分割成两个字符串。

2、

取指定长度的字符串。如在下例中，取最大长度为 4 字节的字符串。

```
sscanf("123456 ", "%4s", str);
```

对于 C++的输入输出就不再详细的讲了，因为 cin,cout 的速度实在太慢，不推荐使用，我一般都是到万不得已时才用。比如当你要读入字符串到 string 对象中时，就只能用

cin 了，这时候还有一个常见的问题，就是如何将一整行字符串读入一个 string 中，这就要用到 getline 函数了。

用法为:

getline(cin, str);

第一个参数就是标准输入流 cin ，第二个参数是接收读入数据的 string 对象，本来还有第三个参数，是结束符的标志，但通常用它默认的就可以了，所以不用管。

注意区分这个 getline 和 cin.getline 的区别：

cin.getline 的用法如下：

char str[20];

cin.getline(str,20); 表示从读入的一行字符串中，取最多 20 各字符放入字符数组 str 中，注意此处的 str 是字符数组，而上面的 str 是 string 对象。

另外需要注意的是，千万不要把 cout 和 printf 混用，因为 cout 是带缓冲的而 printf 不带，所以会使得输出的数据顺序混乱。

下面是几个比较大的在线提交系统（Online Judge）里面有大量历年的竞赛题目，注册一个 ID，然后用自己熟悉的语言（一般有 Pascal/C/C++/Java）写好源代码提交即可，会实时返回信息告诉你是否正确。采用黑箱测试，系统里有一套标准的输入输出数据（对外保密，而且通常数据很多很怪），你的程序的输出和标准输出完全符合即可。常见的返回信息有 AC（Accepted，通过）WA（Wrong Answer，输出有错误）TLE（Time Limit Exceeded，超时）MLE（Memory Limit Exceeded，内存溢出）RE（Runtime Error，发生实时错误）等，只有 AC 了才算做对一题。这里只是一个简要介绍，请大家在做题时先看看各网站上的 FAQ，Enjoy it~~~

浙江大学 Online Judge（ZOJ）http://acm.zju.edu.cn

国内最早也是最有名气的 OJ，有很多高手在上面做题。特点是数据比较刁钻，经常会有你想不到的边界数据，很能考验思维的全面性，现在我主要在这个 OJ 上做题

北京大学 Online Judge（POJ）http://acm.pku.edu.cn/JudgeOnline/

建立较晚，但题目加得很快，现在题数和 ZOJ 不相上下，特点是举行在线比赛比较多，数据比 ZOJ 上的要弱，有时候同样的题同样的程序，在 ZOJ 上 WA，在 POJ 上就能 AC

同济大学 Online Judge http://acm.tongji.edu.cn/index.php

这个 OJ 题数上不能与上两个相比，推荐这个 OJ 的原因是它是中文的，这对很多对英文不太感冒的兄弟是个好消息吧。它也因此吸引了众多高中的 OIer，毕竟他们的英文还差一些呵呵，上面的题目也更偏向高中的信息学竞赛一些。

西班牙 Valladolid 大学 Online Judge（UVA）http://online-judge.uva.es/problemset/

世界上最大最有名的 OJ，题目巨多而且巨杂，数据也很刁钻，全世界的顶尖高手都在上面。据说如果你能在 UVA 上 AC 一千道题以上，就尽管向 IBM、微软什么的发简历吧，绝对不会让你失望的。

俄罗斯 Ural 立大学 Online Judge（URAL）http://acm.timus.ru/

也是一个老牌的 OJ，题目不多，但题题经典，我在高中的时候就在这上面做题的。

UsacoGate Online Judge（USACO）http://ace.delos.com/usacogate

全美计算机奥林匹克竞赛（USACO）的训练网站，特点是做完一关才能继续往下做，与前面的 OJ 不同的是测试数据可以看到，并且做对后可以看标准解答，所以如果大家刚开始的时候在上面那些 OJ 上总 WA 却找不到原因的话，可以试着来这里做做，看看测试数据一般是从什么地方阴你的。

如何设计测试数据

——江苏省常州高级中学 曹文

（文章来源：《NOI 专刊》总第 13 期）

有些同学参加一次信息学比赛之后，自我感觉非常不错，但是测评结果成绩却并不理想。造成这种情况的原因有多方面，但是我认为其中不可忽视的一大原因就是在写完程序之后，他们并不知道如何保证程序的正确性。在这里，我就这个问题提出一点自己的看法。

先从考试的时间分配问题讲起。很多同学觉得考试时间很充分，NOIP 有三个小时，写完 4 个程序绰绰有余。显然这个想法是错误的。考试有三个小时，可这三个小时不是全都用来写程序的。一个合理的规划方案应该是：

1.半个小时读题，想算法；

2.一个半小时写程序；

3.一个小时制作测试数据测试程序。

当然，参加不同的比赛，题目质量不同，身体状态不同，心理压力也不同，这个规划方案不可能按部就班。但我认为它是有一定的指导意义的。下面让我们一起来仔细地分析一下整个方案。

读题和想算法，是比赛的重中之重，想必所有的同学都已经铭记于心，不必再多说了。而写程序的话，也许一个半小时仓促了一点，但是这其实只是一个编程熟练程度的问题。如果我们勤加练习，对算法和各种常用过程（快速排序、堆排序等）了如指掌，这个目标并不难达到。事实上，我校一位在 NOIP2007 中取得满分的同学读题加写程序一共只用了 1 小时 15 分钟。

最后一步，也是本文的中心，就是制作测试数据了。整个规划方案中安排了一个小时用来测试题目，也许有些同学会觉得没有必要。但是我认为，每一道题都应该设计测试数据，有些把握不大的题应该设计多种测试数据以保证测试的全面。因此，在这一个小时里，为 4 个程序设计数据，平均 15 分钟就要设计一个，这还不包括如果你发现了程序中的问题，再修改程序用的时间。这么

一算，一个小时就有点吃紧了，似乎还不怎么够，不过其实还是够的，因为做数据比做题要简单得多。

测试数据分很多种。一般而言可以把它们分成小数据、大数据和极限数据三种。

一、小数据

样例无疑是所有 OIer 的最爱。大家学编程的时候就知道写完程序第一件事就是过样例，样例就是一个典型的小数据。小数据有三大优点：

1.易于调试。很多同学对 pascal 的调试模式有很大的依赖性，静态查错的能力很弱。严格地说这不能算是一个坏习惯，因为调试模式确实给我们带来了高效率，帮助我们在极短的时间找到程序的问题所在。但是调试的弊端就是不能处理大数据量的数据。因此，对于静态查错能力弱而调试能力相对较强的同学而言，小数据很重要，绝大部分的错误都是靠小数据的跟踪调试找到的。

2.易于设计。这一点不用多说。由于数据量小，我们往往可以手工设计质量更高的数据，同时对于数据本身也有直观的了解。与此同时，很多的题都会有所谓的“变态数据”，这和极限数据有着一些不同，它虽然数据量不大，但是剑走偏锋，比如某矩阵题给你一个全都是 1 的矩阵之类的。这种狡猾的数据在评测的时候往往并不罕见，由于这样那样的原因，我们就栽了跟头。为了使得自己的程序更加强壮，我们需要预先测试自己的程序是否能够通过这样的数据。这种变态数据只能够由我们手工设计，因此一般都是小数据。

3.覆盖面广。对于很多题目而言，测试数据理论上存在无穷多组；但是如果有 n<5 并且所有数都小于 10 的限制，那么数据的个数就变得有限了，不妨设是 1000 组。我们可以通过写一个程序，直接把这 1000 组小数据全部都制作出来，然后逐个儿测试。虽然这些数据的数据量小，但是由于它们把小数据的所有可能的情况都包括在其中了，因此你的程序的大部分问题都能够在这 1000 组数据中有所体现。同时，因为是小数据，程序可以在很短的时间内运行出解，例如是 0.05 秒，这样，1000 组数据，也不过只要 50 秒，完全可以接受。但是要注意，生成所有数据的同时，我们还要写一个效率差，确保正确的程序来验

证结果的正确性。因此这种想法至少需要 2 个程序。（具体操作流程参考“大数据”部分）。

二、大数据

大数据是属于那种数据量比小数据大，同时可以使用较弱的替代算法得到结果的数据。一般的操作流程是这样的：先写一个随机化的制作大数据的程序；然后写一个针对题目的效率较差但是正确性能够保证的使用替代算法的程序；最后使用一个批处理文件，进行多次对比测试，即生成一个数据，然后再比较两个程序的结果。一定要注意这三个程序的文件输入输出和批处理的实现，这些地方很容易出错。

大数据的使用方法和前面讲过的小数据的穷举方法差不多，但是相比之下有些许不同：

1.数据量不同。数据量变大之后，对程序是一个新的挑战，一些更加难以发现的问题可能会显露出来。

2.可以随机化。与小数据不同，由于大数据的数据个数过多，不能够穷举完成，因此推荐使用随机化。而随机化显然比穷举要容易编写得多，因此大数据的实现更加方便。而随机化的缺点是，变态数据未必能够随机到。

而与极限数据相比，大数据的优点是可以使用替代算法。极限数据往往不能使用替代算法，因为替代算法往往不能在几秒钟，几分钟甚至几个小时内得出解。

因此大数据是最值得提倡的，我认为如果条件允许，每一题都应该用大数据来测试一下，确保正确性。

三、极限数据

在很多人的观念里，极限数据是非常重要的一种数据。我发现很多同学在通过样例之后第一个测试的数据就是极限数据。我认为这是一种非常不好的习惯，因为极限数据并没有我们想象的那么重要。我们能够通过极限数据了解到什么呢？无非是我们的程序是否会超时，外加我们的程序是否会越界。事实上一些简单得随机化得到的极限数据或者手工出的带规律的极限数据甚至未必真

的能够给我们翔实的信息，因为就算对于这些数据我们的程序能够快速得到结果，我们仍然不知道我们的程序是不是真的不会在处理别的数据时超时或者越界。所谓的极限数据，其实不过是用来测试你的数组有没有越界而已，这才是它的最大的用处。

此外，极限数据容易给你一些虚假的信息。例如你测试一个 1000*1000 的全是 1 的矩阵，结果程序在 0.01 秒内出解了，结果无疑是正确的。于是你得出你的程序不越界、不超时、不错误的结论。一些没有经验的同学甚至可能就不再检查这个程序而努力去完成下一道题。然而这么一个特殊的数据，其实什么问题都说明不了。

因此你又不得不使用特殊数据，因为如果你使用了一个随机的数据，你又不能够确定自己的程序结果是不是正确。

当然，我并没有宣扬以后大家不要再设计极限数据。极限数据仍然有它的重要性。但是它的设计应该是最后一步，在至少设计并测试大数据，确保了程序的正确性之后，再检查程序的极限情况下是不是会超时或越界。一般的测评数据中，极限数据的个数并不会很多，我们不应该因小失大。

虽然对测试数据的种类有了一定的了解，但是很多同学也许对如何测试仍有疑惑。测试程序的原始方法是全部手工的，大致可以概括为：手工运行测试数据生成程序、手工运行源程序、手工运行替代算法的程序，最后目测两个程序的结果是否一样。这个方法虽然直观，但是有很多不可忽视的缺点。比如效率太低，操作时容易有较大误差等。我在这里详细地介绍一下使用批处理的方法。假设我们需要测试一个程序 sample.exe，它的输入输出文件分别是 sample.in、sample.out，那么首先我们制作一个数据生成程序 samplemaker.exe，然后写一个用替代算法的程序 sample2.exe，注意这个程序的输入文件是 sample.in，但是输出文件是 sample2.out。最后写一个 txt 文件：

Samplemaker.exe

Sample.exe

Sample2.exe

Fc sample.out sample2.out

Pause

然后把这个 txt 文件的后缀改成 bat，就变成了一个批处理文件了。双击运行这个批处理文件，就可以自动测试，得出两个程序的结果是否相同了。Fc 命令是用来比较两个文件是否相同，pause 表示暂停，以便让我们能够看清楚结果。

但是这个批处理只能够测试一次，我们需要运行它 n 次来进行 n 次的测试。这个效率仍然偏低。提高效率的方法是，在批处理文件中加入循环语句，如下：

:loop

Samplemaker.exe

Sample.exe

Sample2.exe

Fc sample.out sample2.out

Pause

Goto loop

这样的话，这个批处理就能够运行无数次，每次你只需要敲一下键盘就可以了。这样的效率已经很高了，你需要做的不过是盯着 fc 的结果，找到一个不一样的就关掉批处理，找到 sample.in，然后调试。

不过，对于有些题目来说，这样的效率仍然不够，1 秒可能只能测试几个数据。更好的方法应该是，把 pause 删除，然后把 fc 的结果输入进文件，然后写一个程序判断 fc 的结果，如果发现结果不一样，就把 sample.in 记录下来。如下：

:loop

Samplemaker.exe

Sample.exe

Sample2.exe

Fc sample.out sample2.out >result.txt

Judge.exe

Goto loop

其中，judge.exe 需要我们编写。它应该从 result.txt 中读入数据，判断结果，如果不一样，就进入死循环（例如使用 while true do;语句）。这样你就能够一眼看出有问题出现了，关掉批处理，然后检查 sample.in。

下面我就 NOIP2007 的 4 道题，给大家讲一讲我校部分学生设计测试数据的体会。

一、count 统计数字

输入一个数 n（n≤200000）和 n 个自然数（每个数都不超过 $1.5*10^9$），请统计出这些自然数各自出现的次数，按顺序从小到大输出。输入数据保证不相同的数不超过 10000 个。

本题的算法是很简单的，排序+统计。这里推荐使用快速排序，因为快速排序是众多排序中运行效率最高的一个。对于本题，可以设计三种数据：

1.小数据：小数据应该设计几个变态数据。例如 10 个 0，或者 9，8，7，6，5，4，3，2，1 之类。主要是测试一些不容易随机得到的情况。

2.大数据：本题的代替算法有多个。比较简单的算法应该是冒泡排序或者使用数组下标排序之类。这里推荐使用数组下标排序，因为对于本题的要求（统计次数），这个排序有巨大的优势（编程复杂度很低）。

3.极限数据：由于本题的数据量很大，有必要测试一下程序是否会越界。

二、expand 字符串的展开

我们可以用减号对连续字母或数字进行缩写，于是字符串 a-dha3-68 就可以展开为 abcdha34568。

输入三个参数 p1，p2，p3，再输入一个仅由数字、小写字母和减号组成的字符串（长度不超过 100），请按参数展开此字符串。

各个参数的意义如下：

参数 p1=1→所有填充的字母都写成小写；

参数 p1=2→所有填充的字母都写成大写；

参数 p1=3→所有填充的字母和数字都用星号代替；

参数 p2=k→同一个填充字符连续写 k 遍；

参数 p3=1→顺序填充；

参数 p3=2→逆序填充。

另外，如果减号两边的字符一个是数字一个是字母，或者减号右边的 ASCII 码没左边的大，则该处不变。

本题是比较简单的模拟，关键是要看清楚题意。本题不适合出大数据或者极限数据，推荐使用静态查错的方法。当然，也可以设计一些有针对性的小数据。

三、game 矩阵取数游戏

一个 n 行 m 列的矩阵，每次你需要按要求取出 n 个数，m 次正好将所有数取完。每取出一个数你都会有一个得分，请求出最终的得分最大是多少。

每一次取数的要求：每一行中恰好取一个数，且只能取剩下的数中最左边或最右边位置上的数。

每取一个数的得分：所取数的数值乘以 2^i，i 表示这是第 i 轮取数。

矩阵中的数为不超过 100 的自然数，$1 \le n$，$m \le 80$。

本题是比较简单的动态规划。因为是动态规划，一般很少有较好的替代算法。由于行和列之间没有必然的联系，因此我们完全可以考虑 n=1 的情况。这样的话，可能使用 2^m 的穷举算法，这样 m 可以到 20 左右。因此对于本题，可以设计：

1.小数据：针对一些特殊情况，例如所有的数据都是 0，或者所有的数都是 1 的情况；

2.大数据：使用穷举算法加随机化生成数据，可以测试程序的正确性；

3.极限数据：测试程序是否会超时。

四、core 数网的核

树上的任两点间都有唯一路径。定义某一点到树上某一路径的距离为该点到路径上所有点的路径长度中的最小值。定义树中某条路径的“偏心距”为所有其他点到此路径的距离的最大值。定义树的直径为树的最大路径（可能不唯一）。给出一个有 n 个节点的无根树，请找出某个直径上的一段长度不超过 s 的路径（可能退化为一个点），使它的偏心距最小。请输出这个最小偏心距的值。

题目已经告诉你如下定理：树的所有直径的中点必然重合（这个中点可能在某条边上）。其实这个结论很明显嘛，因为如果中点不重合的话必然可以找到一条更长的路。

5≤n≤300，0≤s≤1000，边权是不超过 1000 的正整数。

相对于前面的三题而言，本题有一定难度。本题告诉了我们很多条件，因此正确的算法有很多，例如 dfs 或者 bfs 加穷举等，这里不作讨论。由于本题与树有关，因此并不容易手工出数据。对于本题，可以设计：

1.大数据：随机化可以出一些大数据，但是一个好的替代算法不容易想到。我觉得穷举还算不错的想法。可以使用 n^2 的时间穷举长度不超过 s 的路径，然后再使用 n 的时间穷举别的点，再用 n 的时间算出距离。这样 n 可以大概到 100 左右，算是不错的算法了。

2.极限数据：出不出都可以，因为只要算法恰当，本题应该不会超时或者越界。

最后笔者还要强调一些问题。

1.不能够做错测试数据。测试数据生成程序时非常容易写的，手工出测试数据也很轻松，但是不能够轻视出数据这一个过程。数据的格式、范围以及是否合法都非常重要。如果你不慎出错了数据，你很可能认为不是数据的问题，

而是程序的问题，然后花费大量的时间检查自己的程序。这无疑是非常不值得的。

2.测试数据虽然是非常优秀的检查程序的方法，但是不能够过于依赖，静态查错仍然非常重要。对于小数据，也许还可以轻松跟踪调试，但是大数据或者极限数据就需要花费大量时间了。因此，笔者着重推荐静态查错的方法。

3.认真看完本文就会发现，对于某一个程序，写三四个小程序来测试它其实一点都不过分或者夸张。因此，打字速度和编程熟练程度就变得非常重要。一个一小时能够写 10 个程序的选手和一个一小时只能写 3 个程序的选手对于算法的理解可能是旗鼓相当的，但是程序的正确率绝对会有天壤之别。冰冻三尺，非一日之寒，勤加练习才是确保在比赛中取得好成绩的根本。