事情的缘起是因为我在写二分查找程序的时候，我考虑到如果数组的长度为1，那么此时r=1，l=0，如果我参考的二分查找的程序正确，那么此时middle=(r-l)/2,刚刚看到这个表达式，然后将1和0带入，我的第一反应middle应该等于1，可是如果middle等于1的话，这就说明我的程序是错误的。为此我写了下面的一个小程序，来检测我的想法是否合理：

#include<stdio.h>

#include<iostream.h>

int main(){

int l=0,r=1;

int middle=(l-r)/2;

printf("middle=%d\n",middle);

middle=(r-l)/2;;

printf("middle=%d\n",middle);

middle=(5-0)/2;

printf("middle=%d\n",middle);

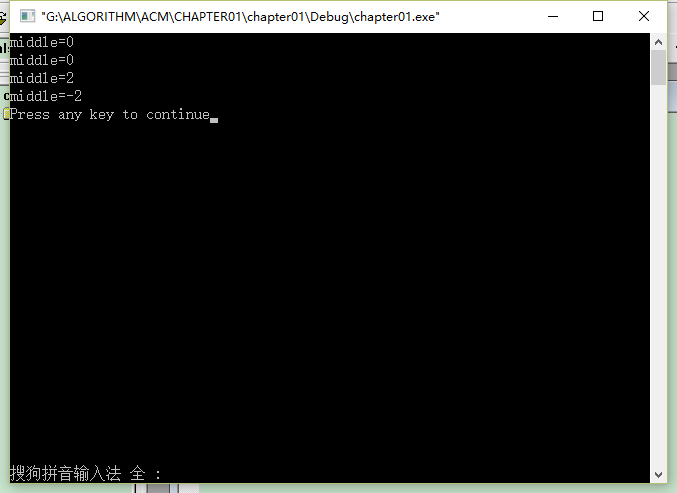
middle=(0-5)/2;

printf("middle=%d\n",middle);

return 0;

}

程序的输出如下：



这就说明：

假设：

int x;

int y;

int z;

int result=(x-y)/z;

设运算结果为result

1. (x-y)/z >0,则采取向下取整，然后将结果赋值给result；
2. (x-y)/z <0，则采取向上取整，然后将结果赋值给result；

采用相反的取整方向可以保证如下的事实：

(x-y)/z+(y-x)/z=0;

因为，如果(x-y)/z和(y-x)/z肯定正负相反，如果采取相同的取整方向的话，即要么都偏大，要么都偏小，这就无法保证两者之和还是为0.所以它们必须采用相反的取整方向来保证相反数这个事实。