软件测试实验说明

实验背景

软件测试是一种使用自动化方式对软件系统进行测试的技术。软件测试中的单元测试对软件中的最小可测试单元（如C语言中的函数）进行检查和验证。穷举整个搜索空间是一种较为盲目的测试方法。掌握被测单元的信息越丰富，就越能进行有针对性的测试。

实验任务

对简单函数`f`进行测试，该函数在文件中以字节为单位，从某两个位置开始（s和t），依次连续读取两个int类型的整数（$s \dots s+3$，$t \dots t+3$），一共读取8个字节，你不需要考虑两个数重叠的情况。不妨设函数`f`读取到的这两个整数为x和y。函数`f`会计算并返回二元函数$g(x,y)=ax+by+c$的值。数据保证$-10,000 \leq a,b,c \leq 10,000$。

选手需要通过构造对应的输入文件（大小为1024字节），通过调用函数`f`与其进行交互，计算得到实数a，b和c的值。需要注意的是，你调用函数`f`的次数不能超过1000次。同时，当你的答案与标准答案的误差小于0.0001时将被认为是正确的。

接口说明

1. 你的代码中需要包含test_function.h的头文件：

```
#include "test_function.h"
```

2. 初始化函数的原型如下：

```
void init();
```

   你需要在程序的一开始就调用初始化函数，以防后续出现奇怪的问题。如果你需要使用调式模式（见第5点），初始化函数应该在`debug_mode`之后，其他代码之前调用。

3. 函数`f`的原型如下：

```
double f (const char *file_path);
```

   其中，函数`f`的第一个参数为你的文件的路径。函数`f`的返回值为$g(x,y)$的计算结果。

4. 提交答案的函数原型如下：

```
bool check_ans (double a, double b, double c, unsigned int random_seed);
```

   其中，函数`check_ans`的前三个参数分别表示你认为正确的$g(x,y)$的系数。第四个参数为你使用的随机种子。如果你没有使用随机种子，请随便传一个数。

5. 为了方便调试，提供了一个接口用于设置函数`f`的随机种子：

```
void debug_mode(unsigned int random_seed);
```

   其中，函数`debug_mode`的参数为你想要设置的随机种子，该函数可以保证随机得到的值是固定的。需要注意的是，**最终运行的时候，你不应该调用该函数**。

6. 运行程序

   下发文件夹`code`中包含了`Makefile`、`test_function.o`和`my_test.c`。在`my_test.c`中包含了一个可以运行的示例（显然这个示例不能找到正确的系数），你需要完成对`my_test.c`文件的重构。

   你可以直接在命令行使用以下指令编译并运行程序：

```
make run
```

报告说明

1. 请详细说明你的策略
2. 请计算你的策略需要调用函数`f`的次数。如果使用了随机化算法，请计算该算法的期望。
3. 请附上三次运行结果的截图，并计算这三次运行的调用次数的平均值。

其他说明

实验得分分为两个部分：运行结果得分（40pts）和实验报告得分（20pts）。

运行结果得分计算方式如下：

$$f_i = \frac{\min\{s_1, \ldots, s_n\}}{s_i} \times 5 + 35$$

$$s_i = \frac{1}{3}(s_{i,1} + s_{i,2} + s_{i,3})$$

其中，$s_{i,j}$表示第i位同学第j次完成的调用次数，f_i表示第i位同学的最终得分。

附录

Ubuntu的存储方式为小端存储，即把低字节放在低地址位。换句话说，对于下面的这段代码：

```
int a = 0x12345678;
```

变量a的第一个字节存储的是$0x87$，第二个字节存储的是$0x56$，第三个字节存储的是$0x34$，第四个字节存储的是$0x12$。示意图如下所示：

a			
1	2	3	4
0x78	0x56	0x34	0x12

而在Windows操作系统上则顺序正好相反。