测试工具

没有装备，怎么打怪。

gdb

不说概念，直接开门见山调一个。

- 调试一个c程序，必须加一个 -g 参数，编译的`可执行程序`才可以被 gdb 调试。

```
gcc -Wall -g index.c -o index
```

- 进入 gdb 命令行模式：
  - 输入 list 列出源代码

```
list
```

  - 再次输入 list简写l或回车 会继续列出后续源代码

```
l 或 直接回车
```

  - 也可以列出`某个函数`名的源代码

```
l some_function
```

  - 开始执行程序

```
start
```

  - 一条一条地执行

```
next/n
```

  - 如何调试另一个函数内部中去

```
start

# 执行到 main 中的调用 some_function 函数的地方时，停住
next

# 用 step命令（简写s），进入 some_function() 函数（执行下一行语句，如有函数调用则进入函数中第一行 ，想想递归）
step

# 继续一行一行走 some_function 的每一行
n
n
backtrace     # 查看栈帧
n
info locals   # 查看局部变量
...

# 查看main主函数的局部变量
backtrace
```

```
frame 1       # 查看栈帧1
info locals   # 先是一堆乱数字，没有问题，因为处于初始化当中

# 然后用 step 再继续走 some_function 一行一行
n
n
n
print var_some_function     # 打印 some_function 中一变量的值!
finish    # 直接走完到 当前函数的return!

# 中途改变变量值
(gdb) set var var_some_function=0
(gdb) finish
# print也可以改变变量值和函数表达式，同上
(gdb) print var_some_function=123
(gdb) print printf("var_some_function=%d\n", var_some_function)

# 跟踪显示。每次停下来的时候都显示 var_some_function 的值
display  var_some_function

# 设置一个断点：
(gdb) list    # 看到源代码行数号
(gdb) break 12  # 为源码12行设置一个断点！（也可以是为函数名设断点！！）
(gdb) continue    # 继续运行（到下一个断点！）【next 和 continue 与断点是最佳使用组合！】
info breakpoints   # （此时用 info 可查已设断点的详细状态！包括命中几次～ ）
# 删除一个断点：
(gdb) delete breakpoints 2    # 每个断点有一编号，可用编号指定删除某个断点
(gdb) info breakpoints    # 再看，已删
# 暂时停用一个断点：
(gdb) disable breakpoints 3    # 日后可重启
(gdb) info breakpoints    # 查看
(gdb) enable 3                  # 启用!
(gdb) info breakpoints    # 查看

# 高级
# 有条件的中断
(gdb) break 3 if var_some_function != 0
(gdb) run     # 以上的设置为基础，从头开始走一遍（自动停在该中断处，先是被中断时的详细信息）

(gdb) x/7b thescanf      # 打印存储单元 thescanf，

# ok
# 注意，以上随时可以使用一下命令查看更多内容：
#
# backtrace （ bt ）  查看栈帧；
# info （ i ） + locals   查看 some_function 函数`局部变量`的值；
#
# 查看 main 函数当前局部变量的值
#   先用 frame 命令（简写为 f ） 并选择1号栈帧；
#   再查看局部变量 info （ i ） + locals；
#
# 打印某个变量值 print
# finish 结束for循环等直接到当前函数终点———当前函数的返回return处
#
# set var 改变变量，就像 firebug 直接改 HTML 一样。如上
# print 也可以改变变量值。如上
#
# display var_some_function 跟踪显示。每次停下来的时候都显示当前 var_some_function 的值
#
# x 命令，可以看到 存储单元 的内容。
```