

**实验报告**



**题目： 熟悉Linux系统及其相关软件环境**

**班 级： 2020211306**

**学 号： 2020211376**

**姓 名： 马天成**

**学 院： 计算机学院**

**2021年 10 月 26 日**

目录

[一、实验目的 3](#_Toc86306686)

[二、实验环境 3](#_Toc86306687)

[三、实验概况 3](#_Toc86306688)

[四、实验步骤 5](#_Toc86306689)

[实验一 5](#_Toc86306690)

[1. 编写源代码 5](#_Toc86306691)

[2. 编译运行 5](#_Toc86306692)

[3. 用gdb进行调试 6](#_Toc86306693)

[4. objdump生成反汇编 7](#_Toc86306694)

[实验二 7](#_Toc86306695)

[1. 源代码 7](#_Toc86306696)

[2. 行列时间比较 8](#_Toc86306697)

[3. 不同优化参数 8](#_Toc86306698)

[实验三 9](#_Toc86306699)

[1. 源代码 9](#_Toc86306700)

[2. 指令使用 10](#_Toc86306701)

[3. objdump反汇编 12](#_Toc86306702)

[4. 调试出现相应步骤 12](#_Toc86306703)

[实验四 14](#_Toc86306704)

[1. 高内存需求 14](#_Toc86306705)

[2. 高时间需求 15](#_Toc86306706)

[五、实验分析 17](#_Toc86306707)

[六、实验总结 17](#_Toc86306708)

[七、诚信声明 18](#_Toc86306709)

# **一、实验目的**

1. 熟悉linux操作的基本操作；
2. 掌握gcc编译方法；
3. 掌握gdb的调试工具使用；
4. 掌握objdump反汇编工具使用；
5. 熟悉理解反汇编程序（对照源程序与objdump生成的汇编程序）。

# **二、实验环境**

简述使用的工具

1. SecureCRT或其他远程登陆工具（服务器：10.120.11.12）
2. Linux
3. Gcc编译器
4. GDB调试工具
5. Objdump命令反汇编

**大1班（1-4班）报告邮寄（最迟时间：2020年11月1日晚23：59）：**

[clavicle@bupt.edu.cn](mailto:clavicle@bupt.edu.cn)

**大2班（5-8班）报告邮寄（最迟时间：2020年11月3日晚23：59）：**

[yangyyj98@bupt.edu.c](mailto:yangyyj98@bupt.edu.c)

# **三、实验概况**

实验内容一（15分）

在linux环境下，编辑课件中源程序（注意程序的完整性）（包含源程序的开发环境截图），采用gcc编译该程序（要求分别采用-o和-O参数，并比较两者性能，编译指令截图），采用gdb进行调试，让程序运行到for函数语句（调试截图），运用objdump工具生成汇编程序（给出main函数的汇编程序截图）

#include<stdio.h>

int main(void)

{

double counter;

double result;

double temp;

for(counter=0;counter<2000.0\*2000.0\*2000.0/20.0+2020;

counter+=(5-1)/4){

temp=counter/1979;

result=counter;

}

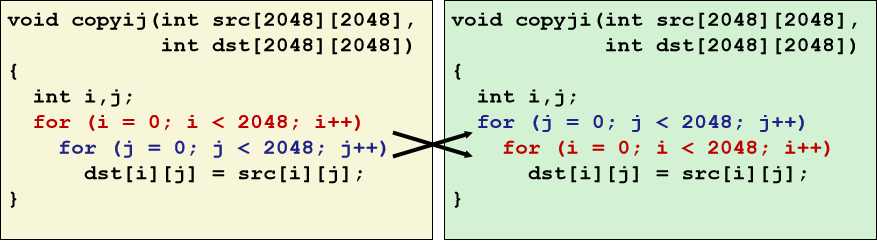
printf(Result is%lf\\n,result);

return 0；

}

实验内容二（15分）

在linux环境下，分别打印输出如下算法所需时间



分别设置不同优化参数，给出运行时间

实验内容三（30分）

现有int型数组a[i]=i-50,b[i]=i+y，其中y取自于学生本人学号2019211x\*y的个位。登录bupt1服务器，在linux环境下使用vi编辑器编写C语言源程序，完成数组a+b的功能，规定数组长度为100，函数名为madd（），数组a，b均定义在函数内，采用gcc编译该程序（不使用优化选项），

使用objdump工具生成汇编程序，找到madd函数的汇编程序，给出截图；

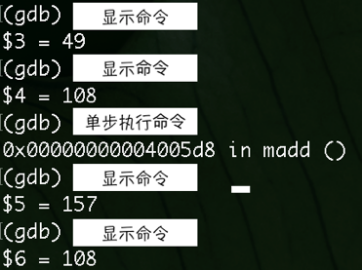
用gdb进行调试，练习如下gdb命令，给出截图；

gdb、file、kill、quit、break、delete、clear、info break、run、continue、nexti、stepi、disassemble、list、print、x、info reg、watch

找到a[i]+b[i]对应的汇编指令，指出a[i]和b[i]位于哪个寄存器中，给出截图；

使用单步指令及gdb相关命令，显示a[xy]+b[xy]对应的汇编指令执行前后操作数寄存器十进制和十六进制的值，其中x，y取自于学生本人学号2019211x\*y的百位和个位。

学号2019211999，a[99]+b[99]单步执行前后的参考截图如下（实际命令未显示出）：



实验内容四（加分项，20分）

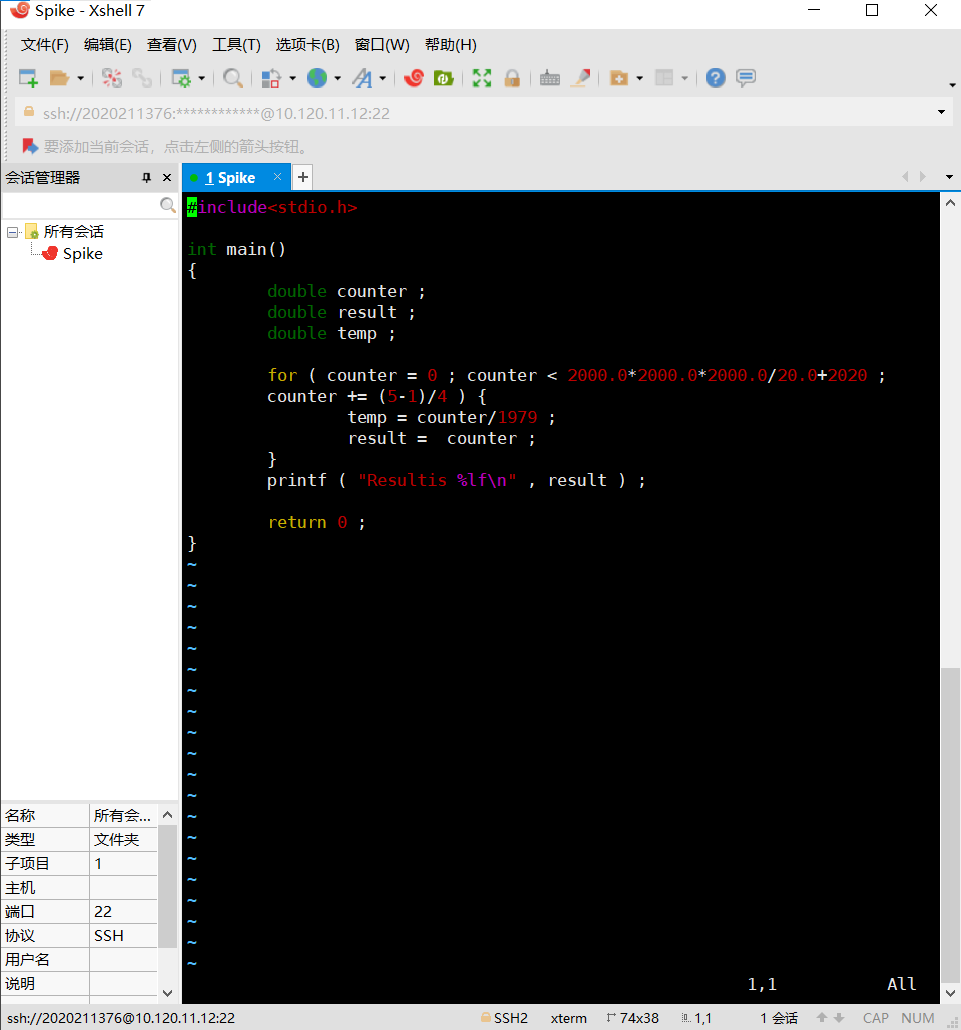
任选高复杂度算法（具体算法自选，类型分为高计算量类型和高内存需求类型2类算法），通过设置不同优化参数，分析算法的运行效率。

# **四、实验步骤**

## 实验一

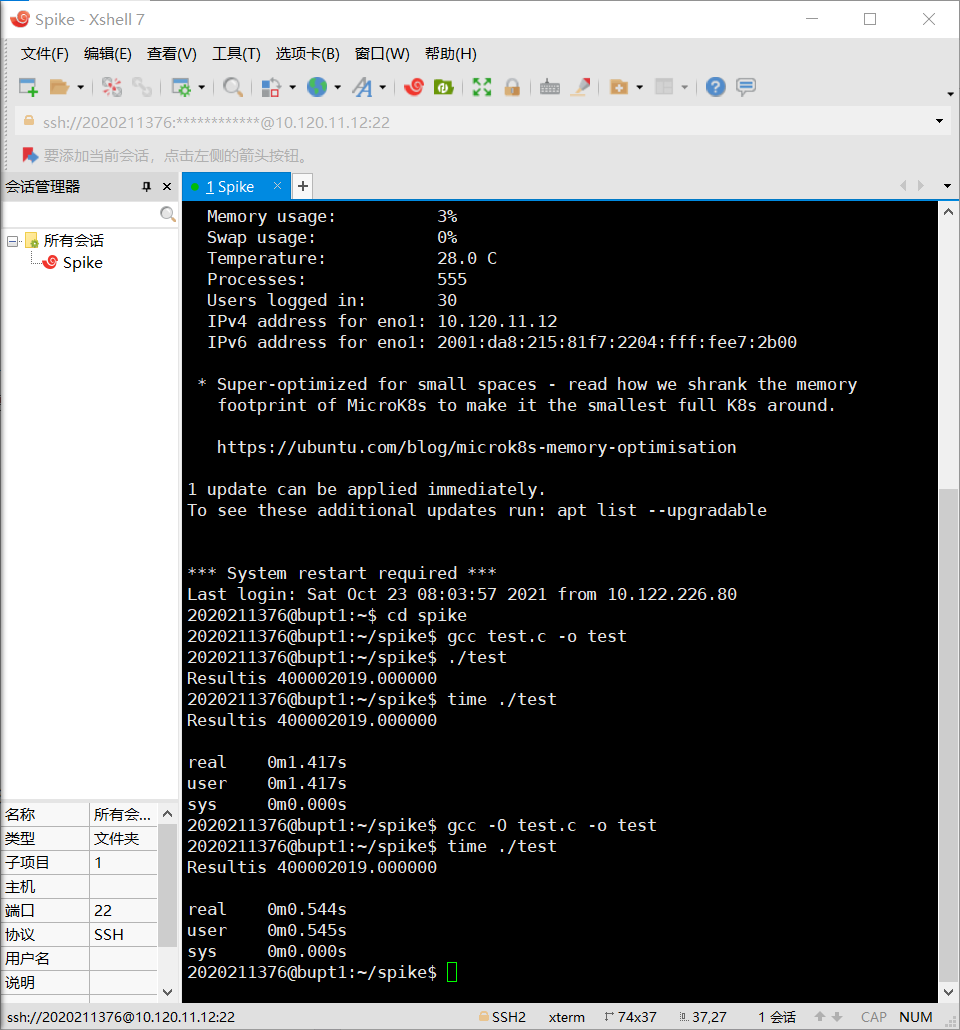
### 1. 编写源代码

源代码截图（老师给的代码有误）



### 2. 编译运行

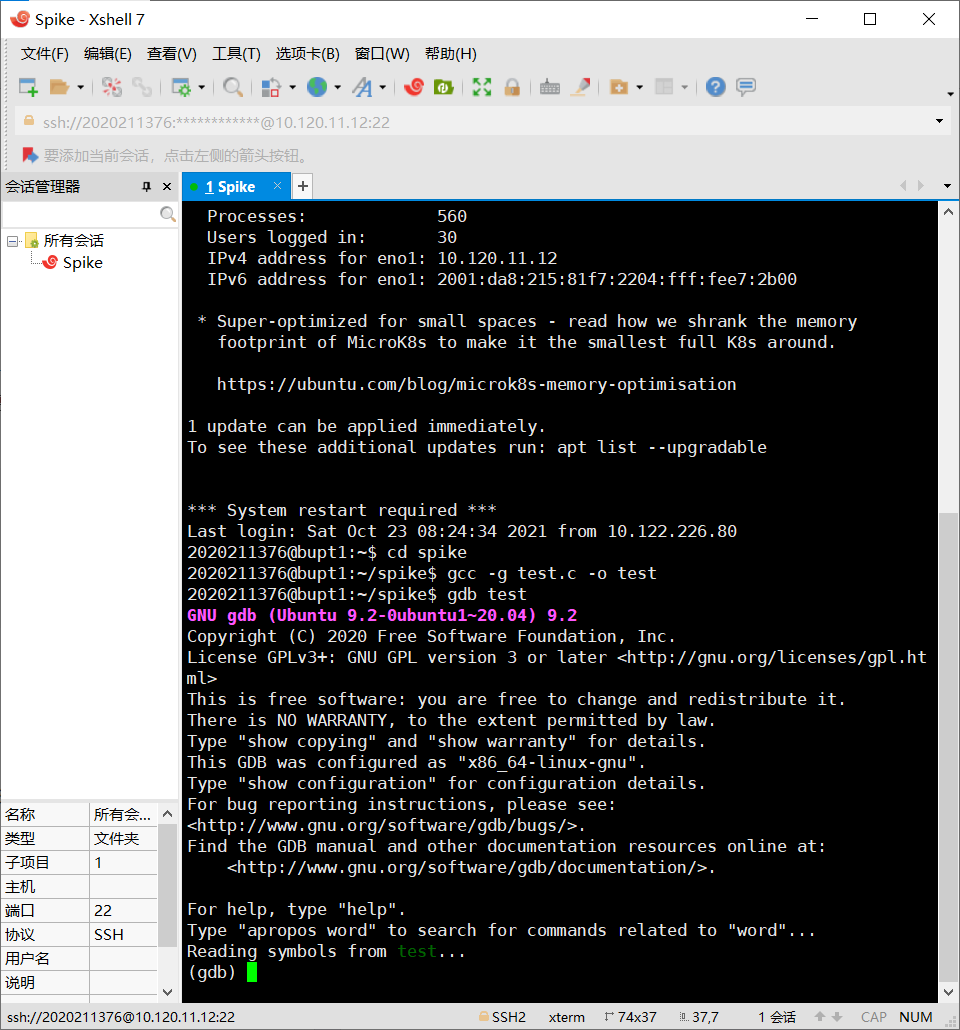
两种参数编译和比较性能( -o -O )



显然，选择编译优化后会快很多。但是不够快，后面优化选项有更快的，这只是基本的优化。

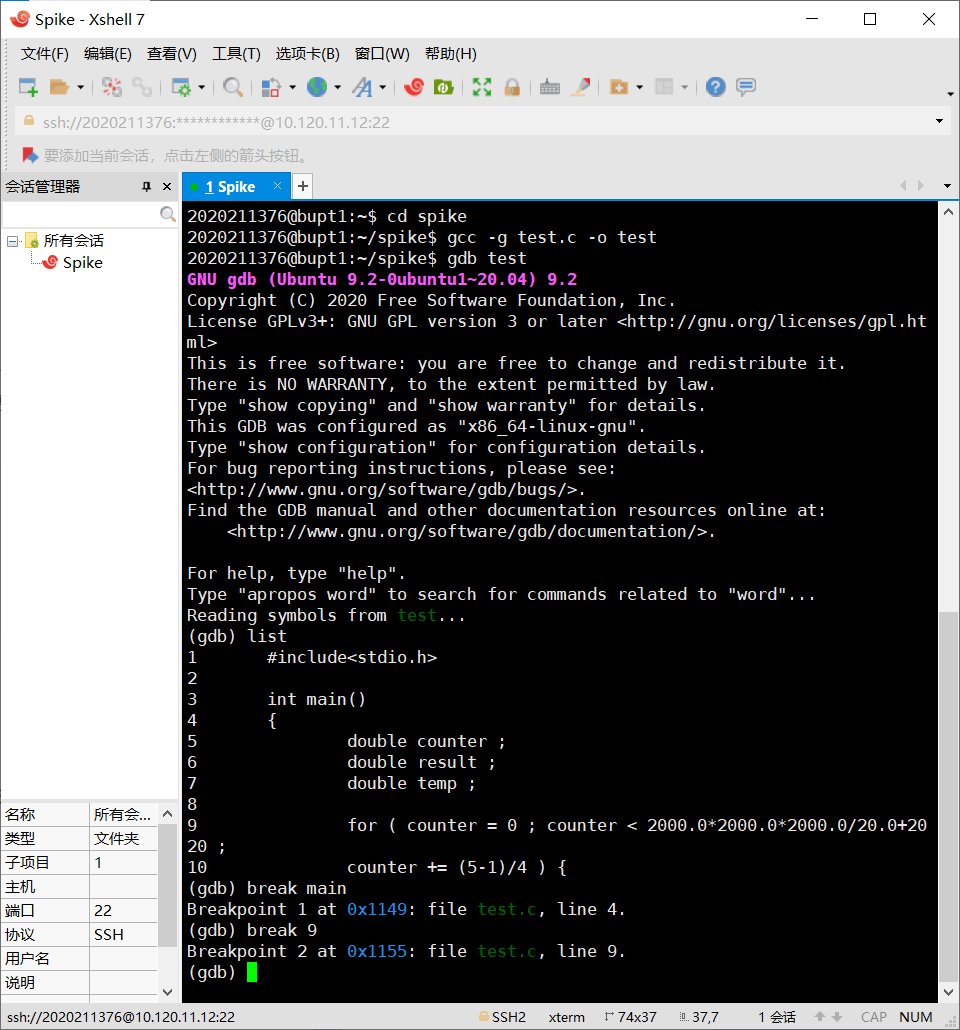
### 3. 用gdb进行调试

先用gcc生成可调试文件，然后用gdb调试可调试文件



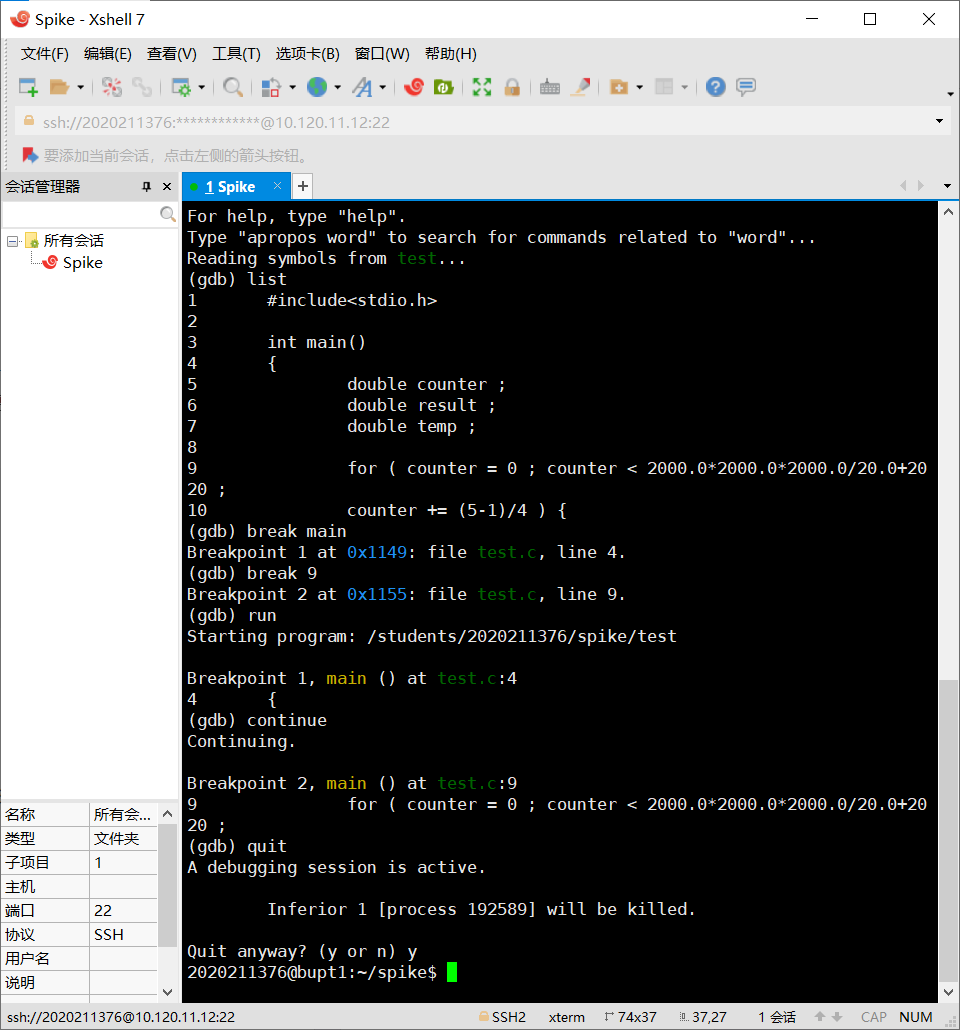
再查看test.c 并设置断点：

先用list查看所在行，再去设置行所在断点



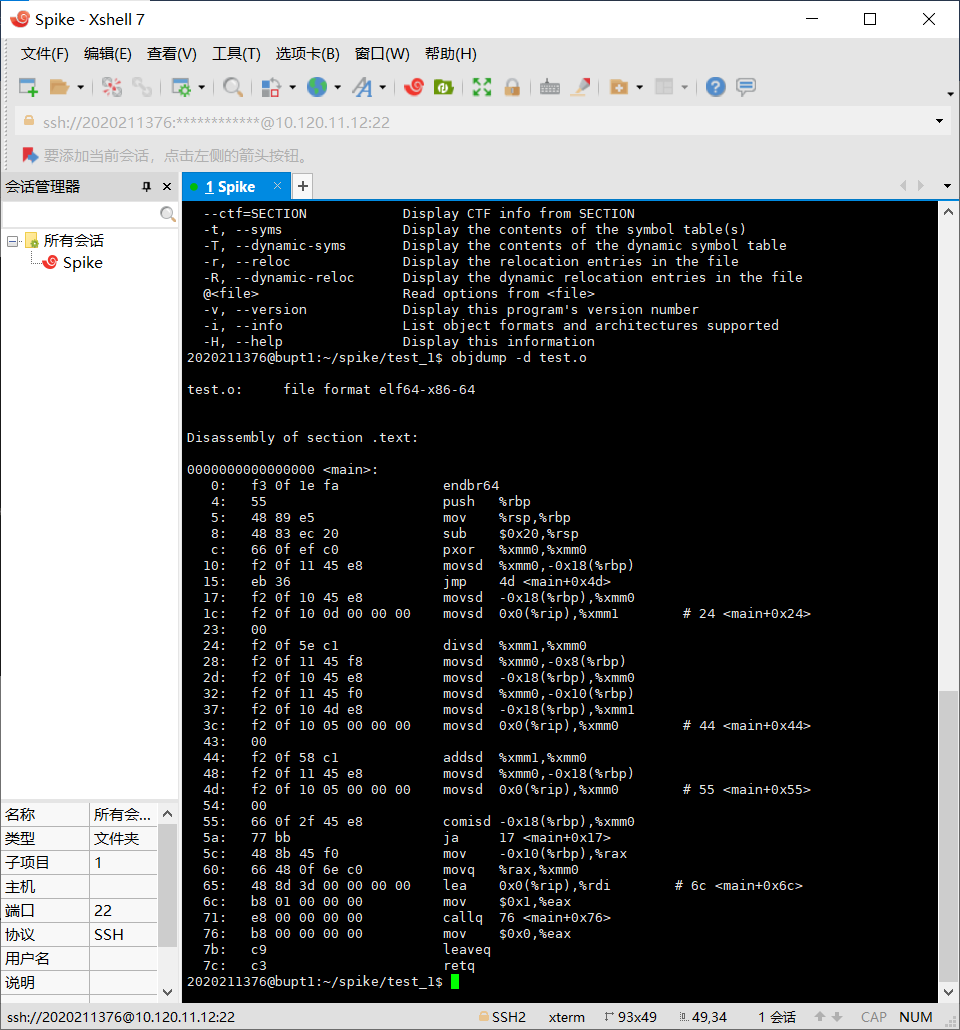
run，调试至for：

run之后到达第一个断点main，然后continue达到第二个断点for



### 4. objdump生成反汇编

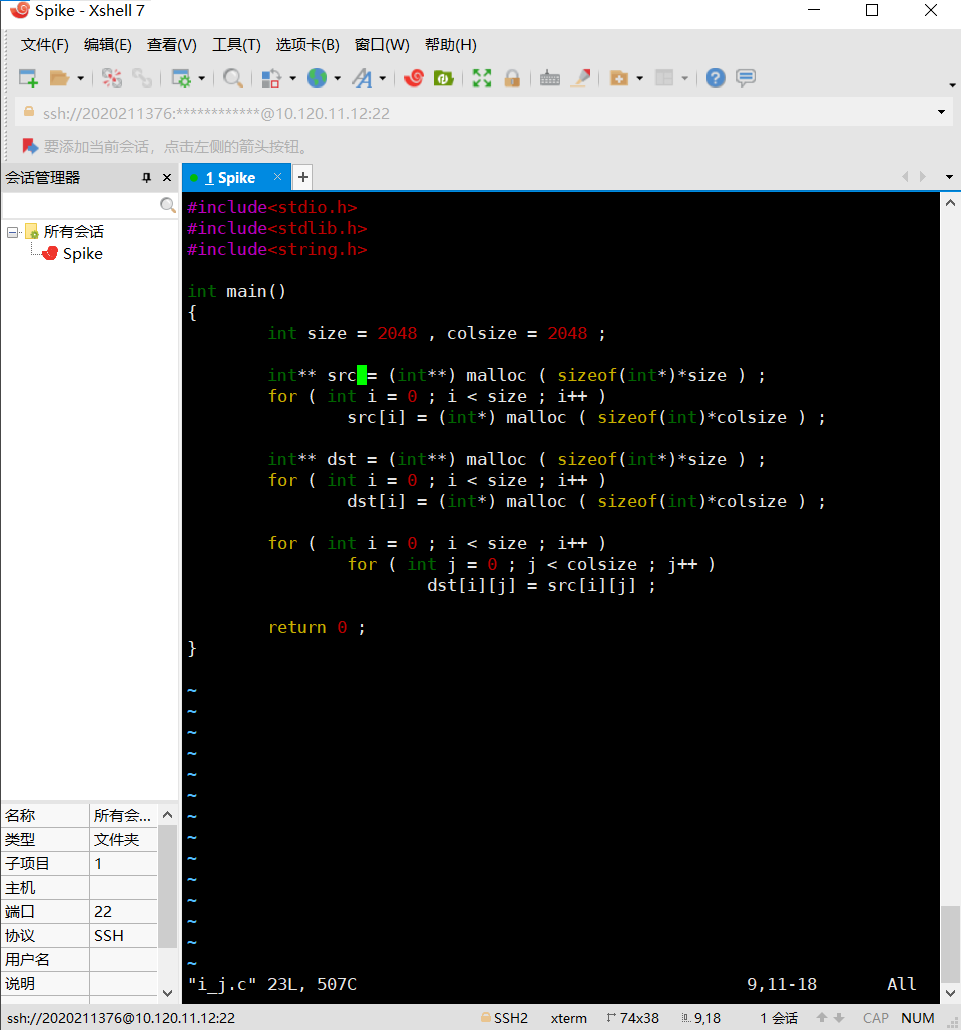
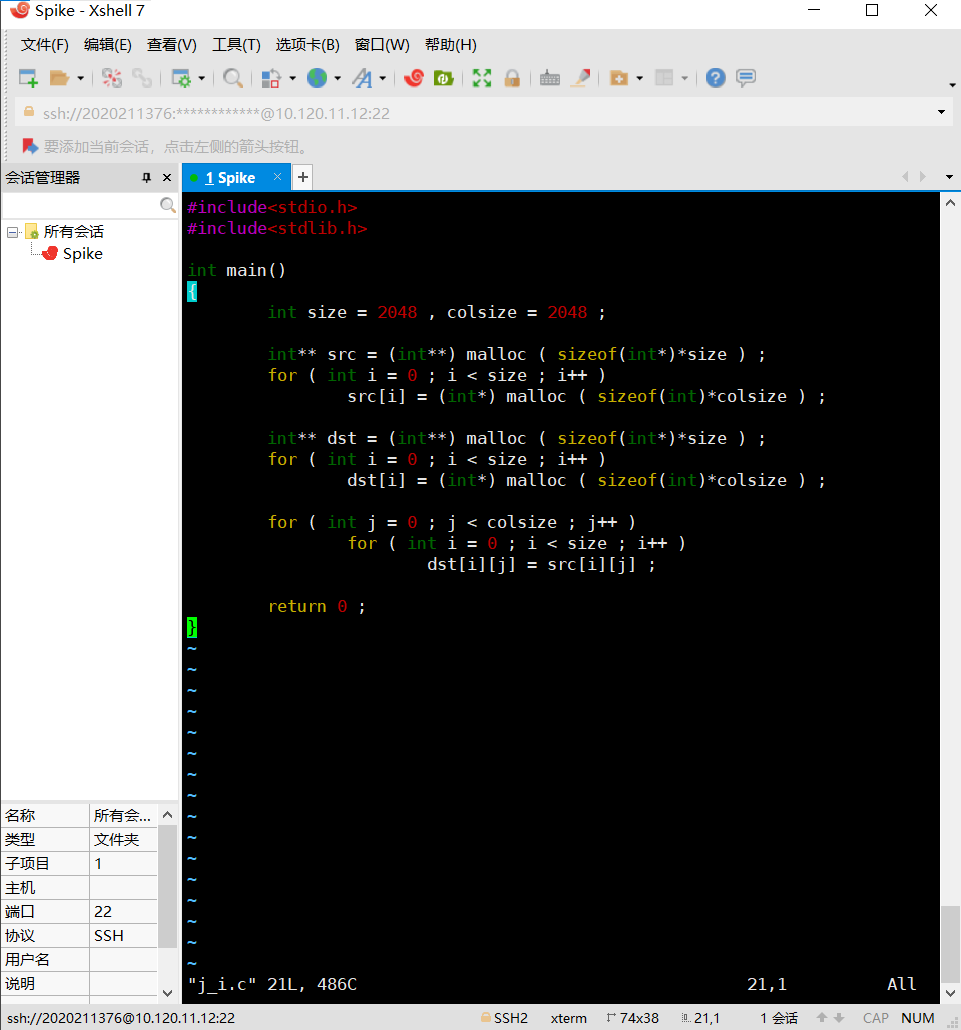
用objdump反汇编.o文件，总共这么长。



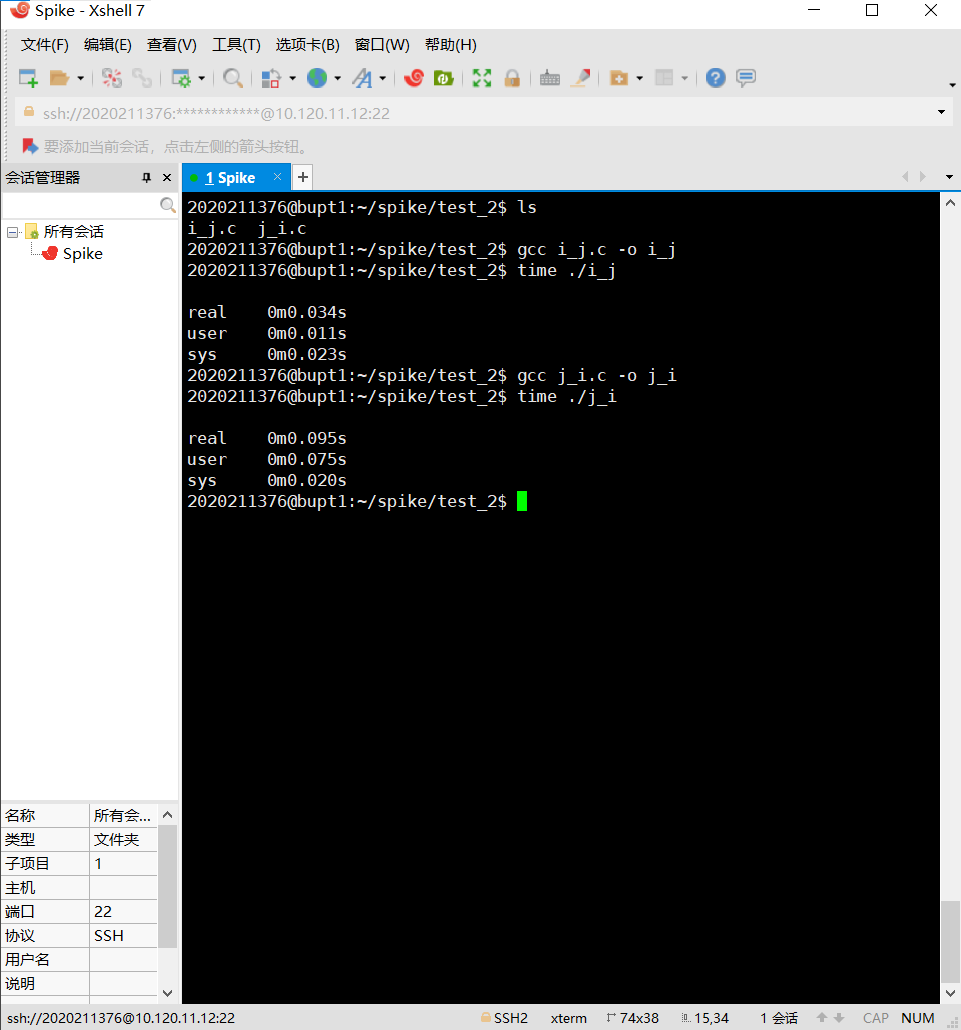
## 实验二

### 1. 源代码

先行后列 先列后行



### 2. 行列时间比较



通过比较知道，我们的先行后列比先列后行快很多。

因为文件是以流的形式输入，先列后行无需跳转，所以快很多。

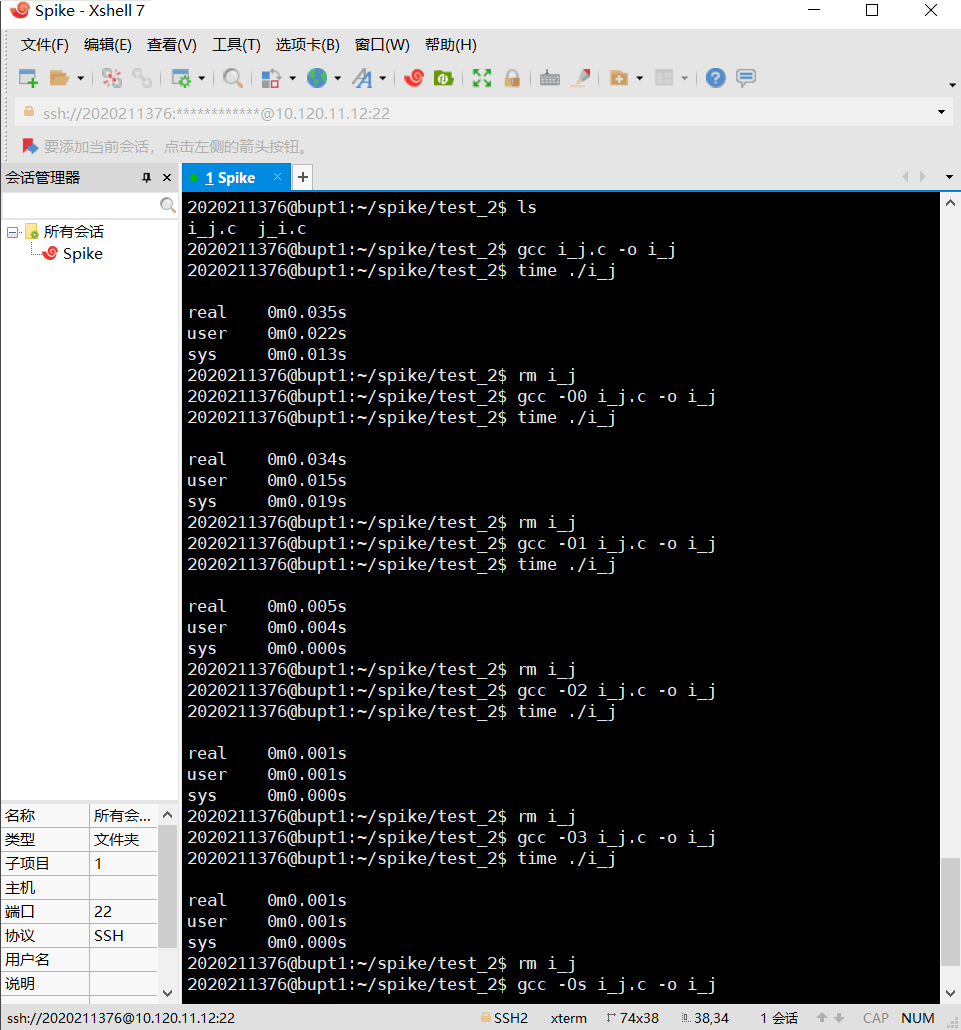
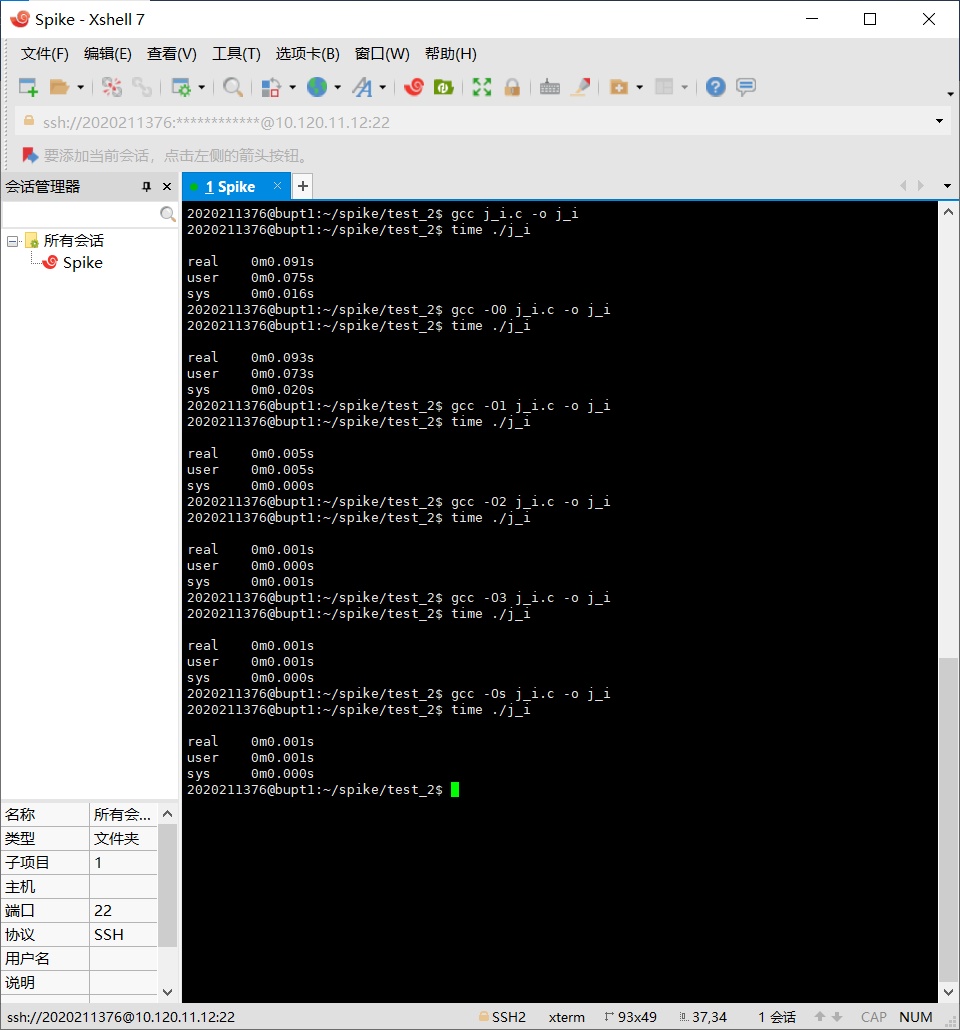
相邻的存储结构的工作会快一些，就和顺序存储和链式存储一样，顺序的操作往往会快一些，因为索引的代价会小很多。

所以这告诉我们，我们在写程序时要考虑存储结构的问题，在空间和时间上找到一个最好的平衡点，写一些高质量代码，比如算法优化和结构体变量顺序优化等，来减小时间复杂度。

### 3. 不同优化参数

分别测试了( -o -O0 -O1 -O2 -O3 -Os )

i\_j.c j\_i.c



-O0： 不做任何优化，这是默认的编译选项。

-O和-O1： 对程序做部分编译优化，对于大函数,优化编译占用稍微多的时间和相当大的内存。使用本项优化，编译器会尝试减小生成代码的尺寸，以及缩短执行时间，但并不执行需要占用大量编译时间的优化。

-O1：优化会消耗少多的编译时间，它主要对代码的分支，常量以及表达式等进行优化。

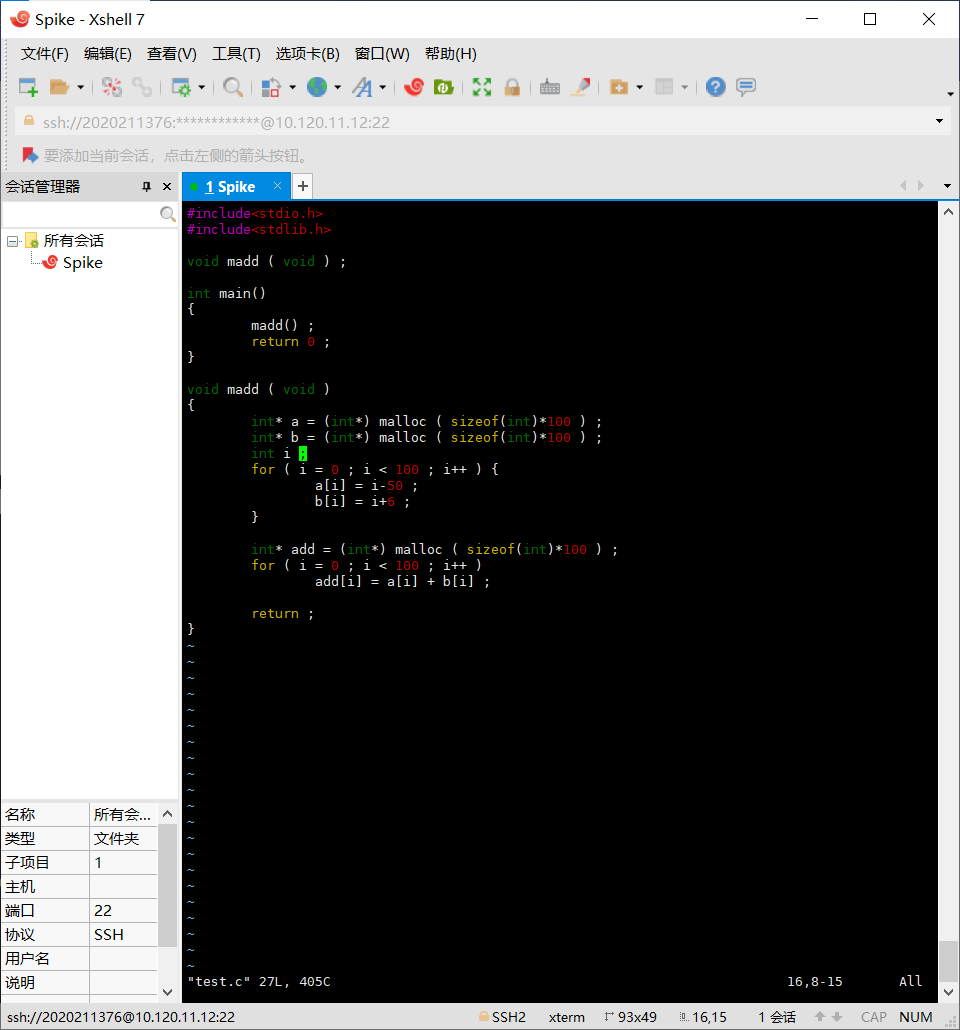
-O2：会尝试更多的寄存器级的优化以及指令级的优化，它会在编译期间占用更多的内存和编译时间。

-O3： 在O2的基础上进行更多的优化。例如使用伪寄存器网络，普通函数的内联，以及针对循环的更多优化。

所以，我们要根据实际代码选择不同的优化参数，像空间复杂度大的问题，就可以用O3来进行优化。

## 实验三

### 1. 源代码



先分配空间，然后进行赋值操作（不关心赋值是什么，只关心操作复杂度）

所以我只用malloc分配空间后就执行赋值操作就可以了。

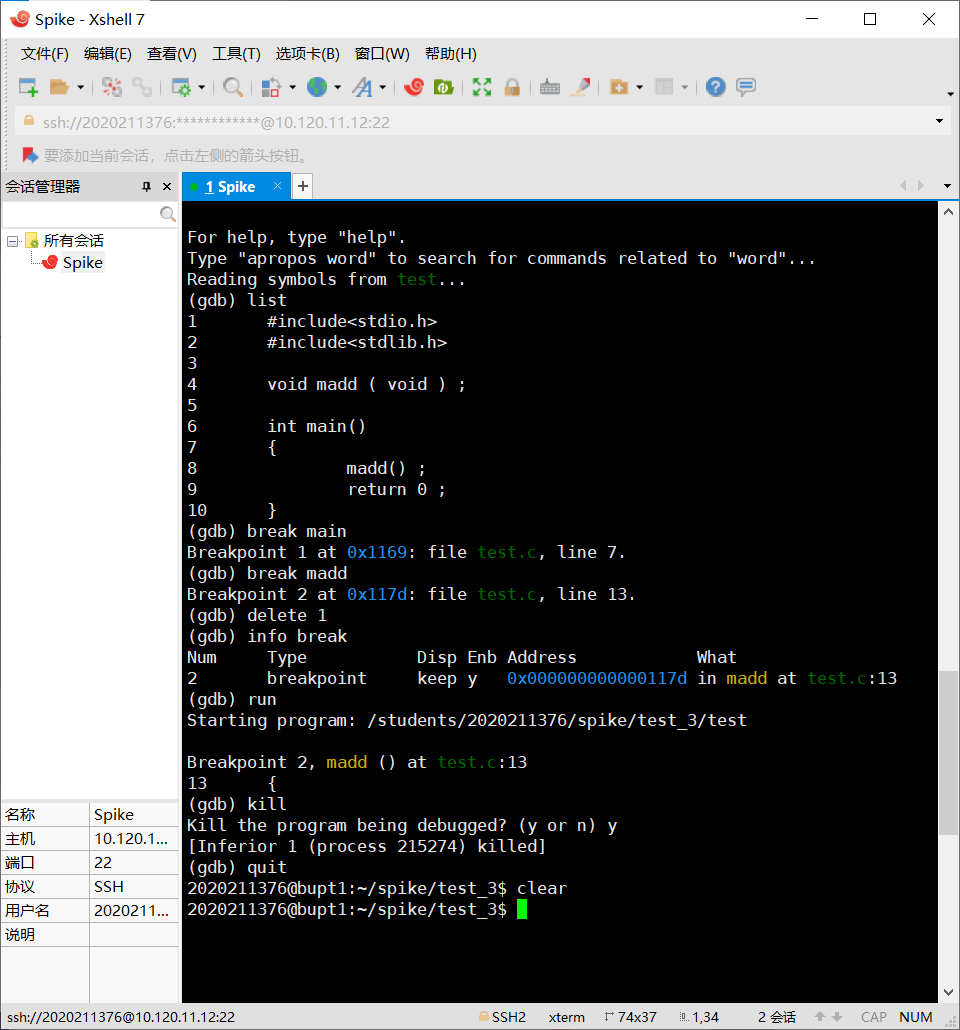
注：我的学号是2020211306，所以x=3，y=6.

a[i] == -14 b[i] == 42 为要调试到的位置。

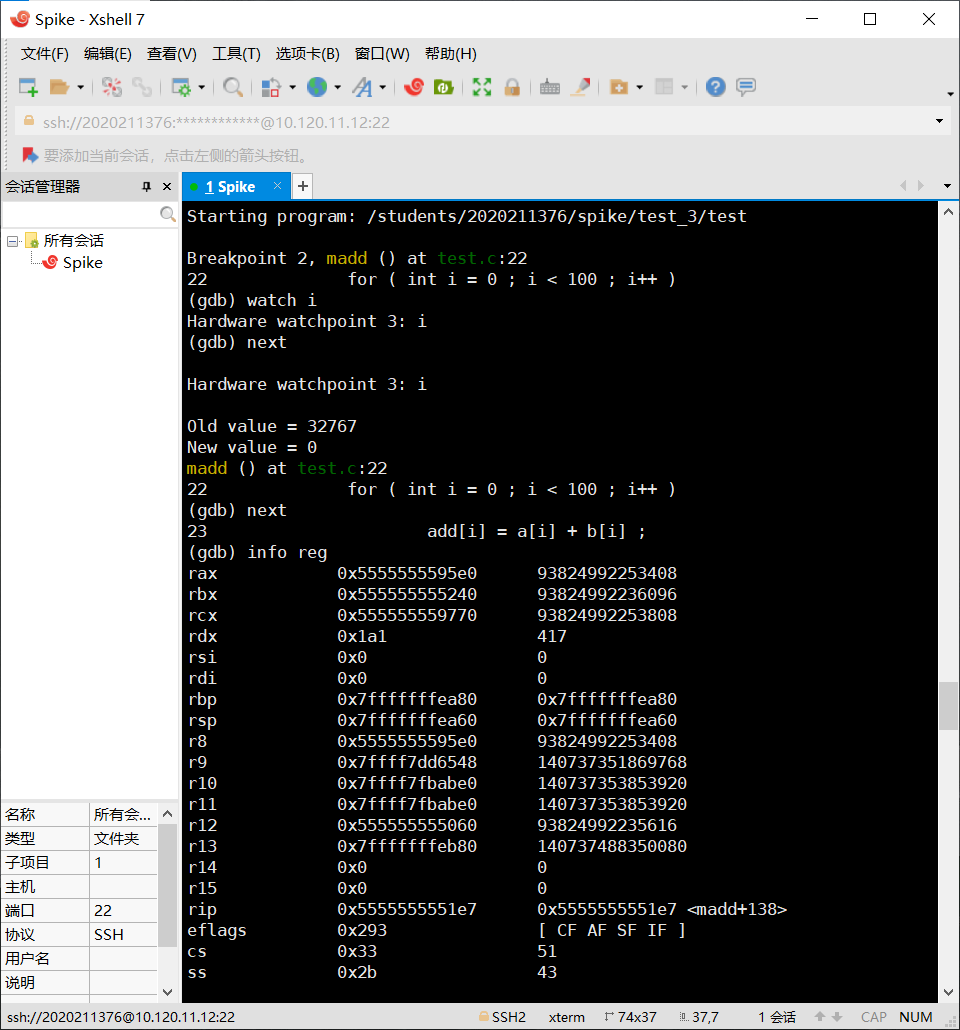
### 2. 指令使用

因为一个一个截图不容易看，而且为了表明使用场景，我就设计了几个流程来展示常见命令：

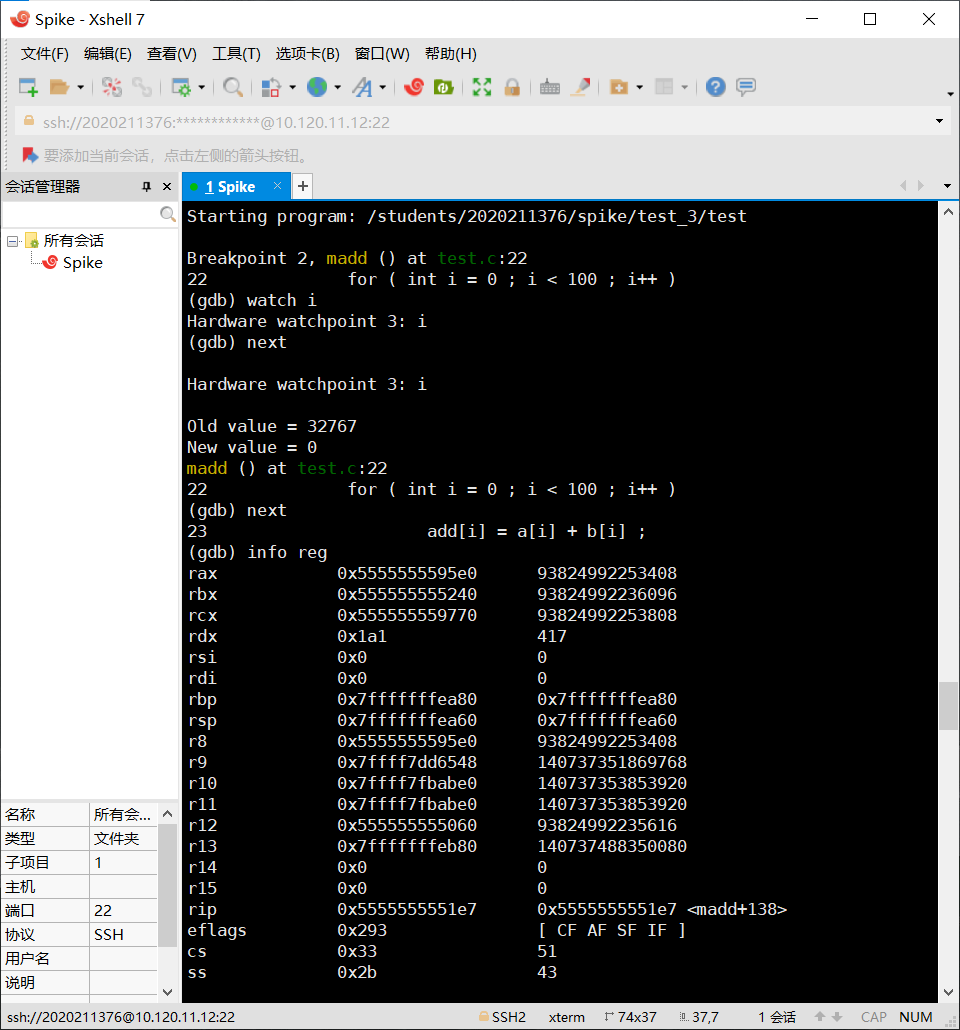
list\_break\_info break\_delete\_run\_kill\_quit\_clear



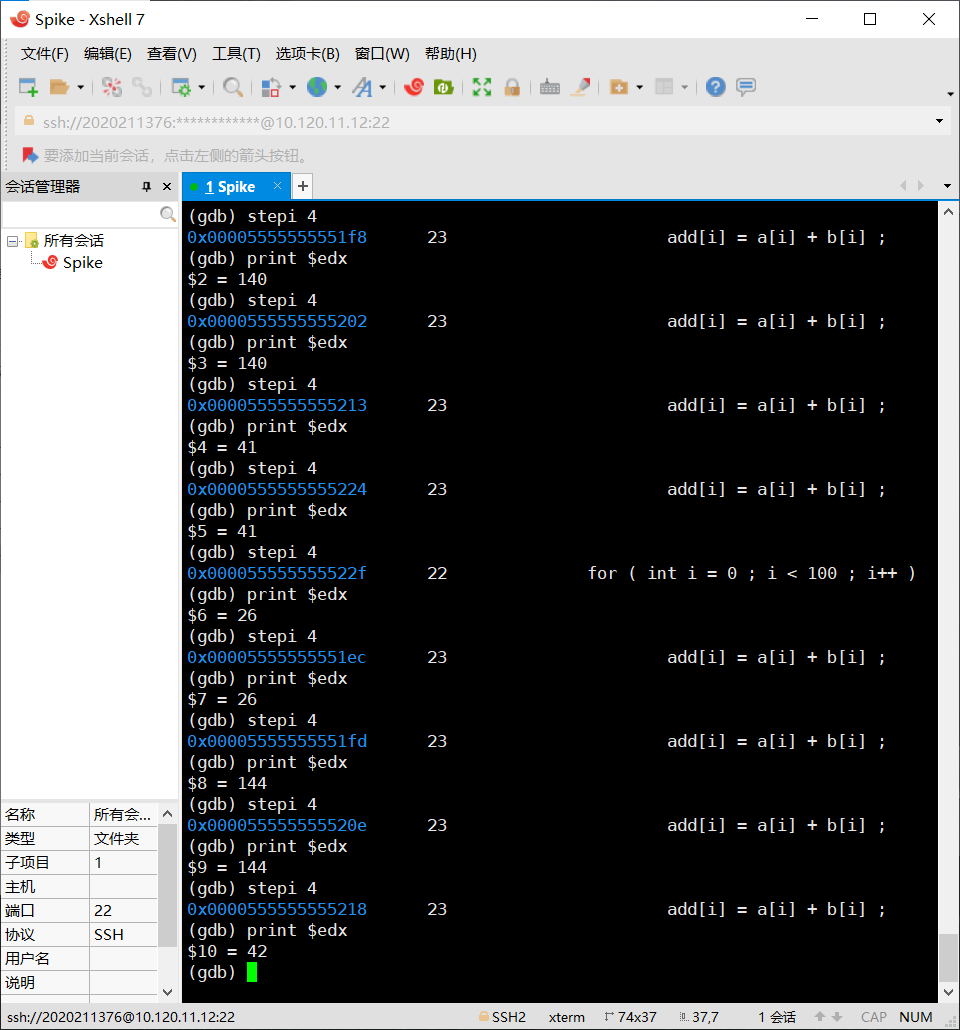
watch\_next



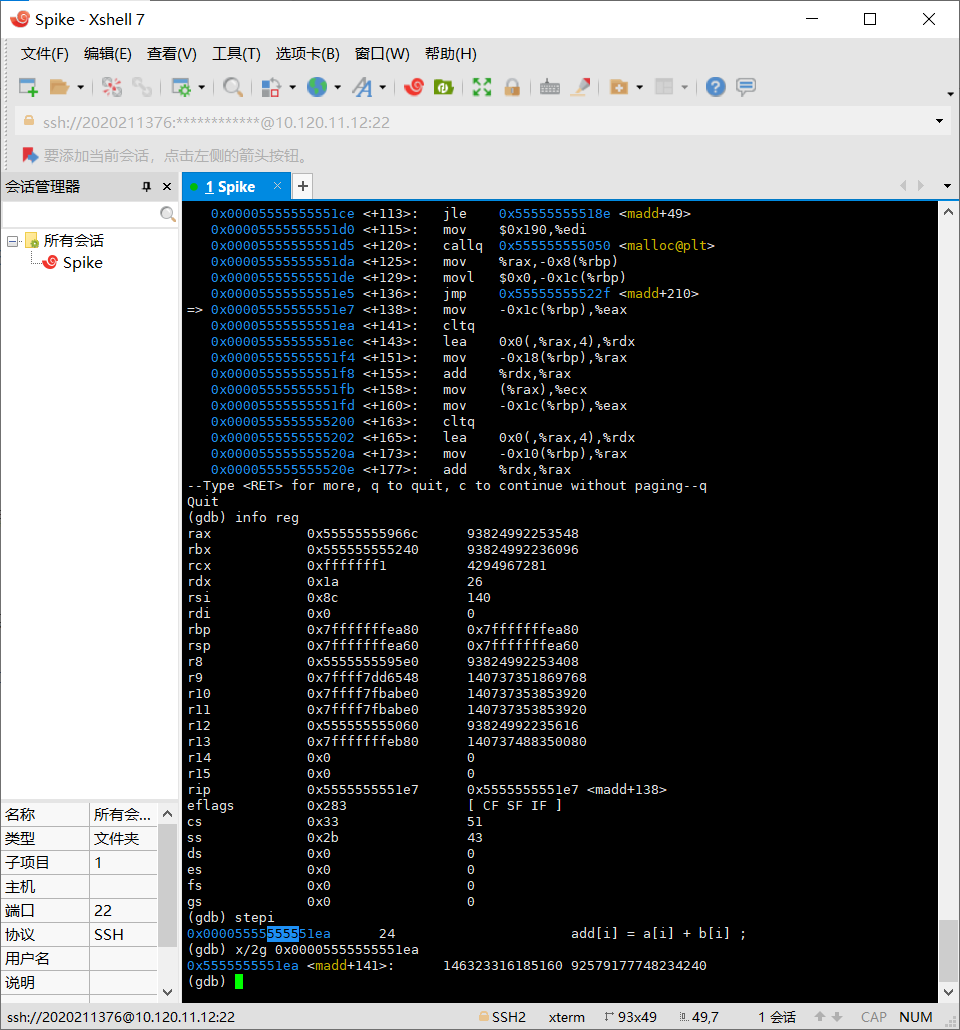
info reg (要求达到的地方的寄存器值)



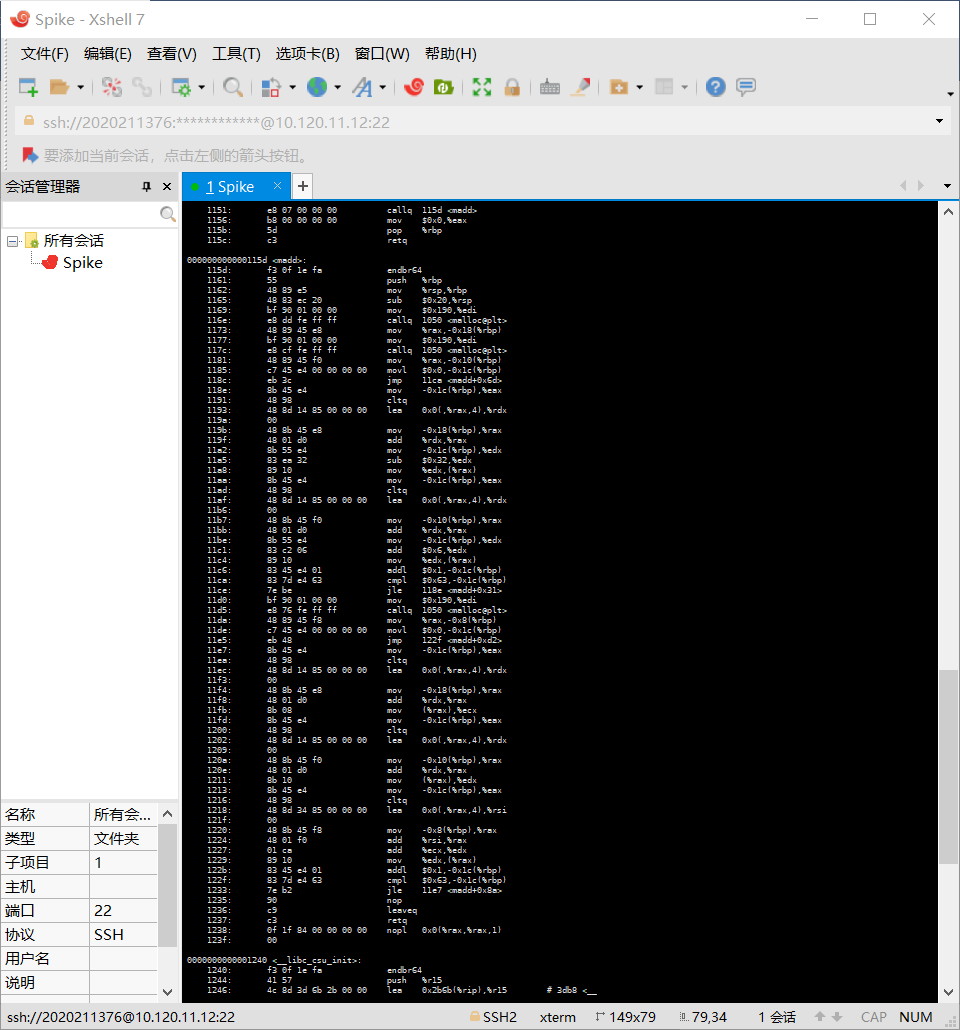
print $edx\_stepi

（其实这一步已经是下面的单步调试后出的答案了，但后面有更详细的）

disassemble\_stepi\_x



### 3. objdump反汇编



### 4. 调试出现相应步骤

本人学号为2020211306，即需要达到的下标值为36。

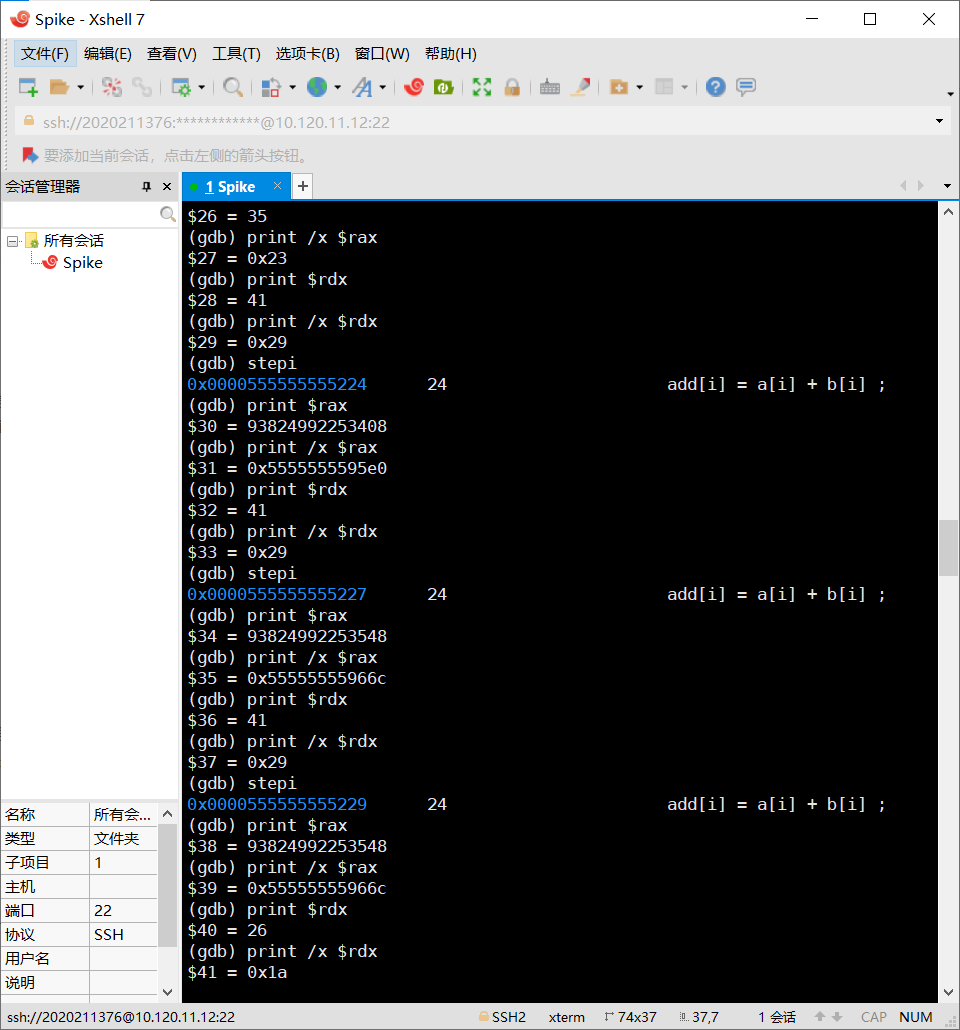
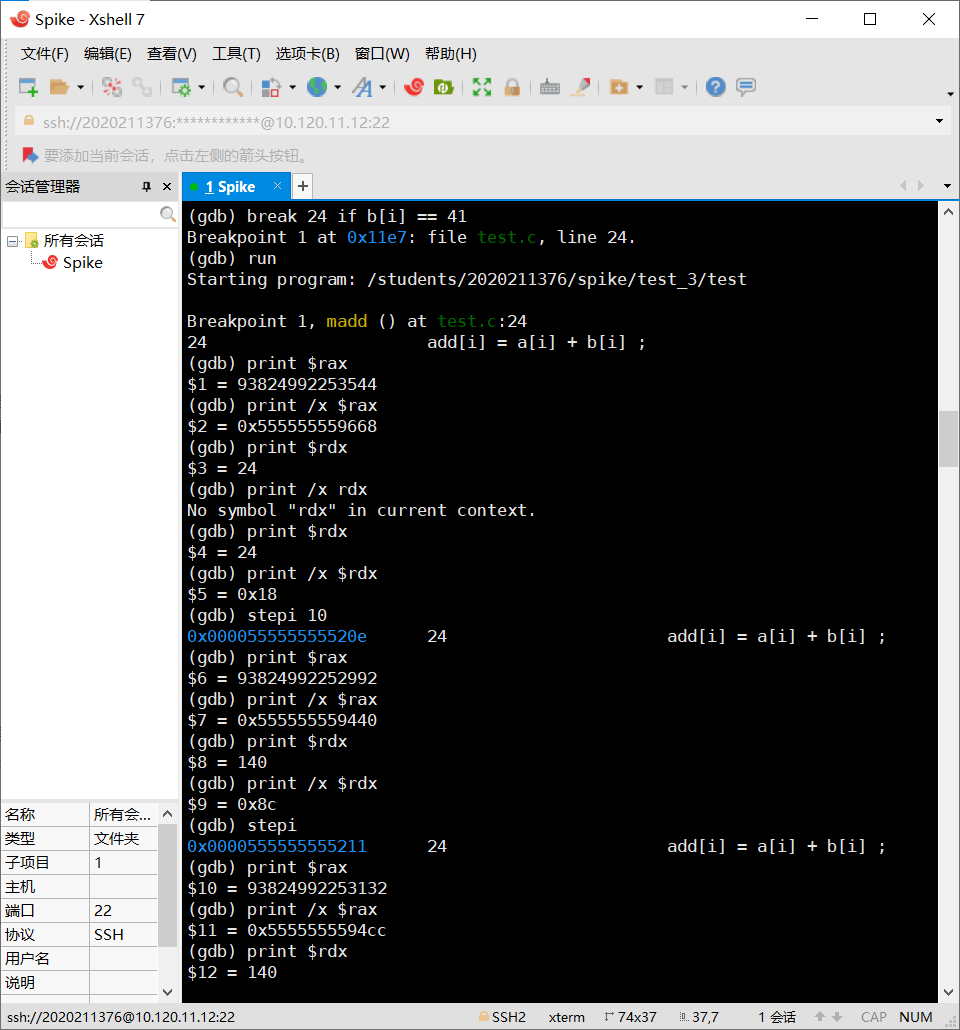
经过观察disassemble，我的加法寄存器是：rax rdx (即64位寄存器)。

1. 逐步stepi，

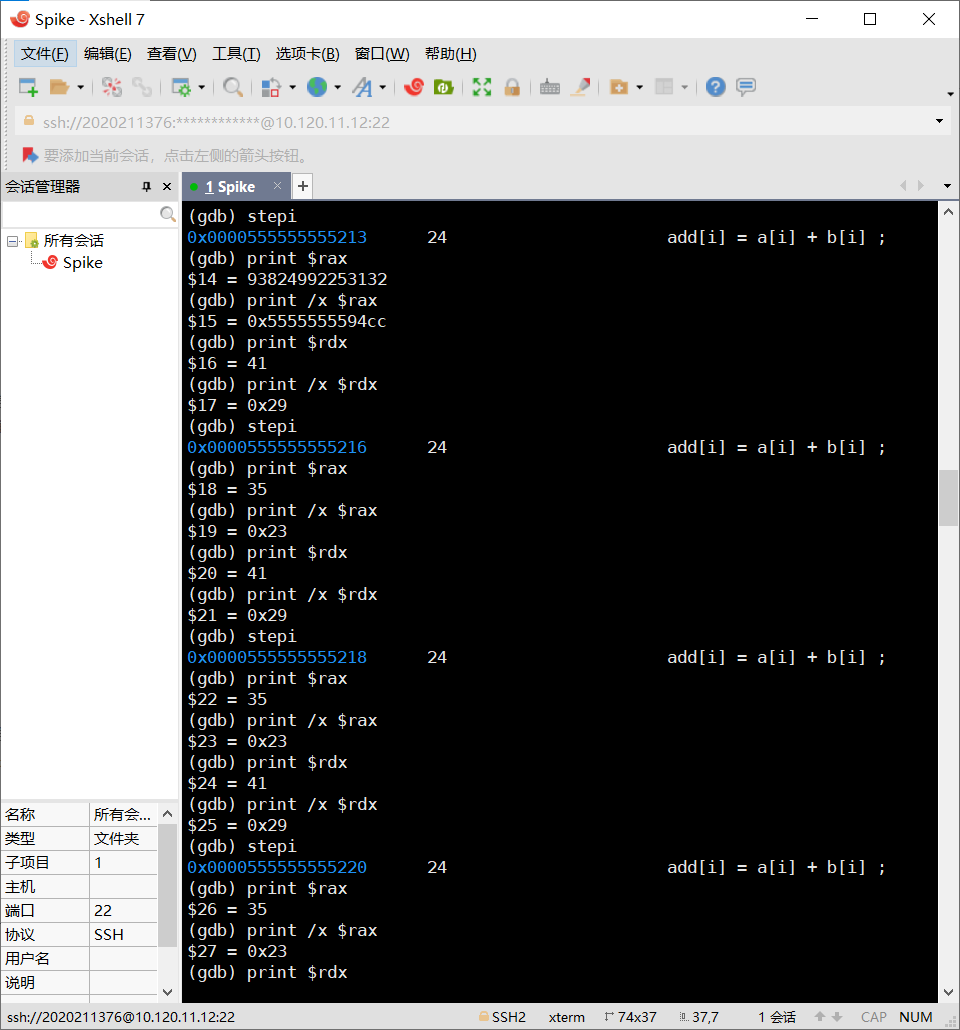
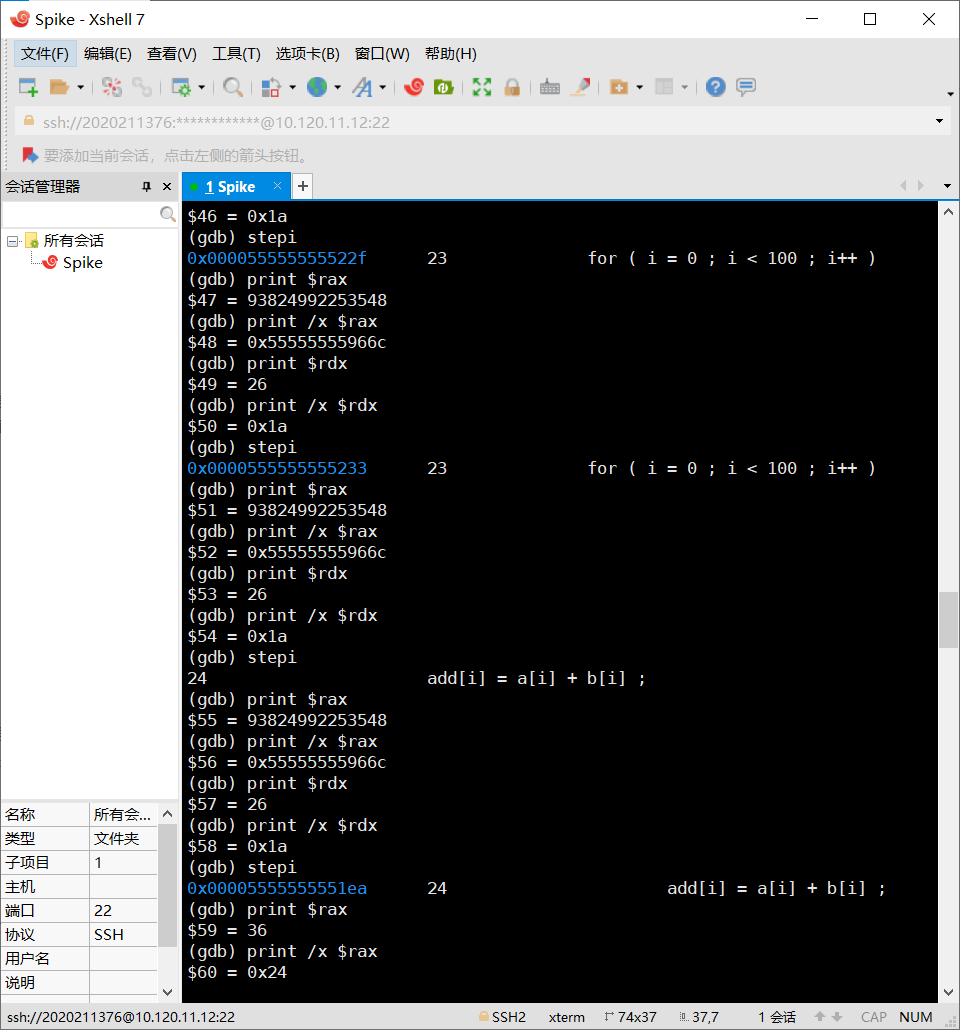
2. 打印rax和rdx十进制和十六进制的值，

3. 观察完整个循环。

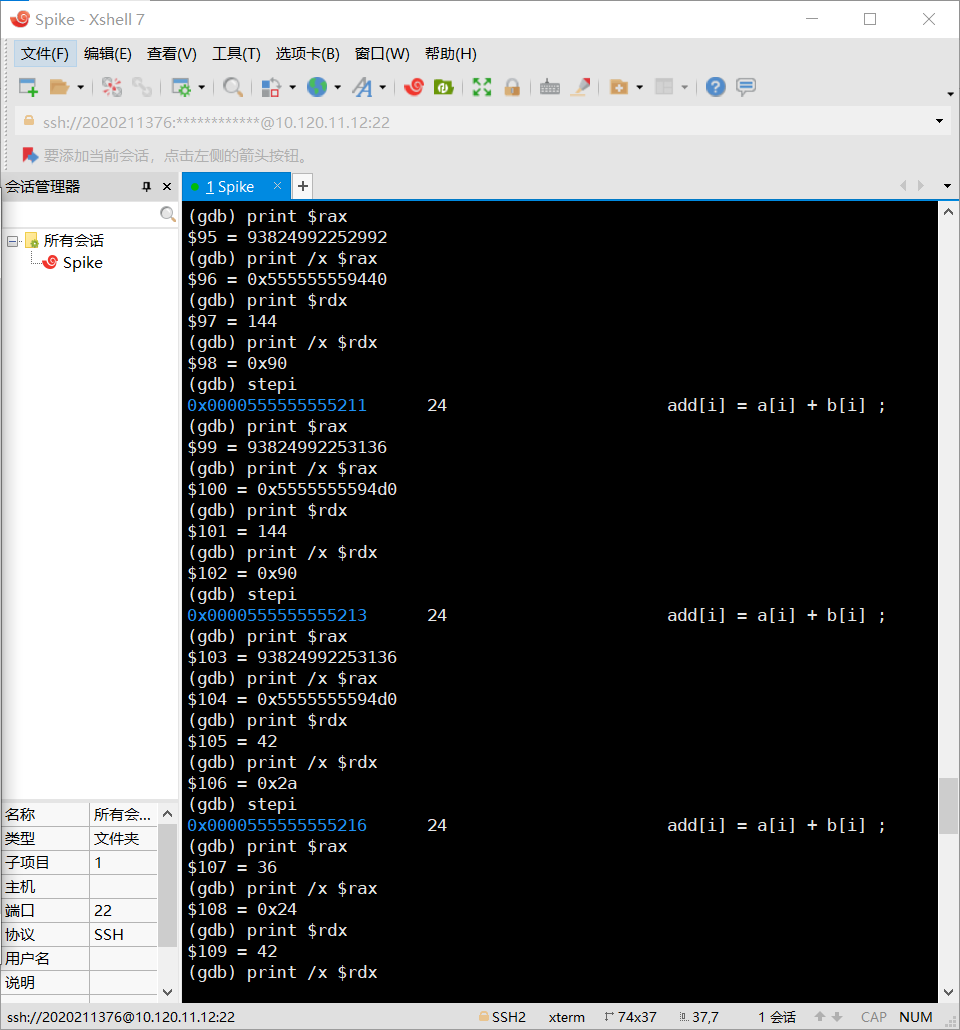
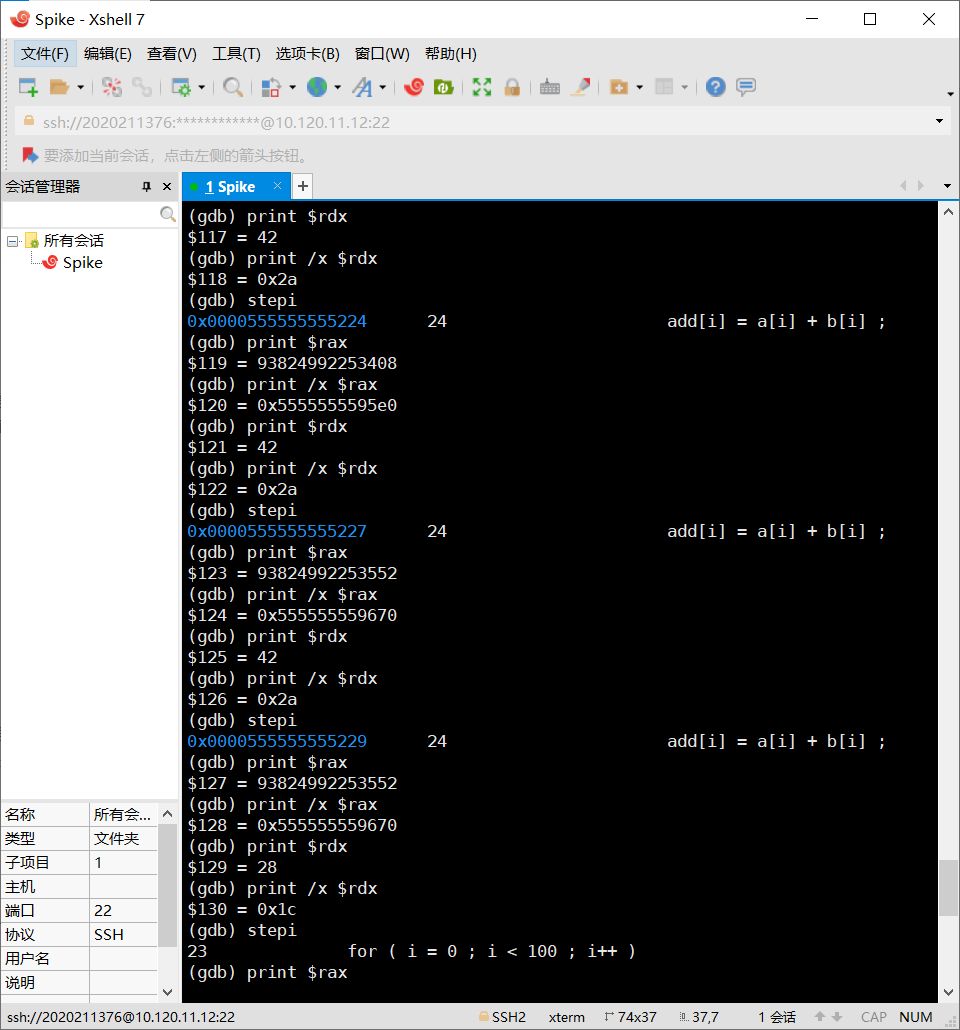
开始出现35，马上到达36： 还是35：



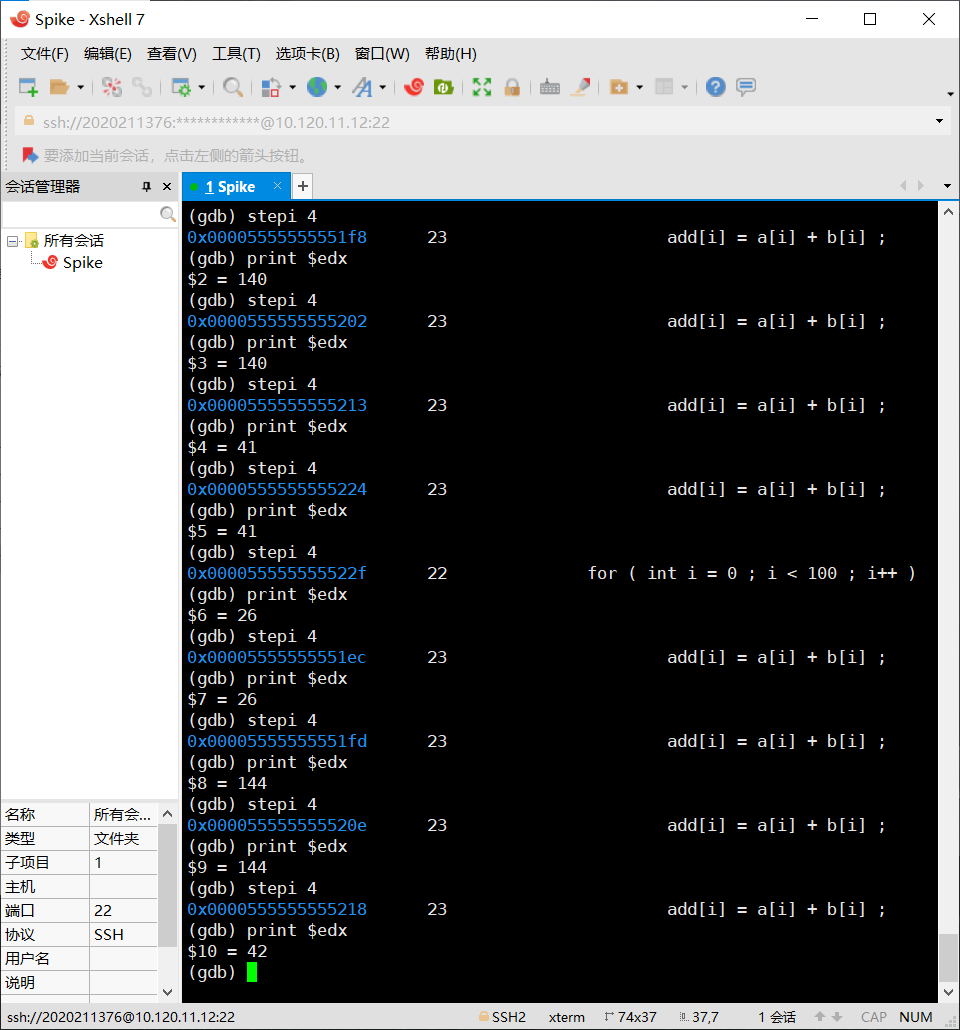
出现35和41（b[i]）： 开始出现36：



同时出现36（b[i]）和144: 出现42和28，并且结束循环：



如果想要简单的结果，那么我有stepi 4调试的输出：



注：41 42是b[i]的值，其中出现了140、144、26、28都是寄存器的临时值。

其中，i=36一直在rax寄存器。

在此过程中，我经历了很多困难，从不会调试到设置断点，从逐步调试到快速调试，从不会看寄存器到会看寄存器。

## **实验四**

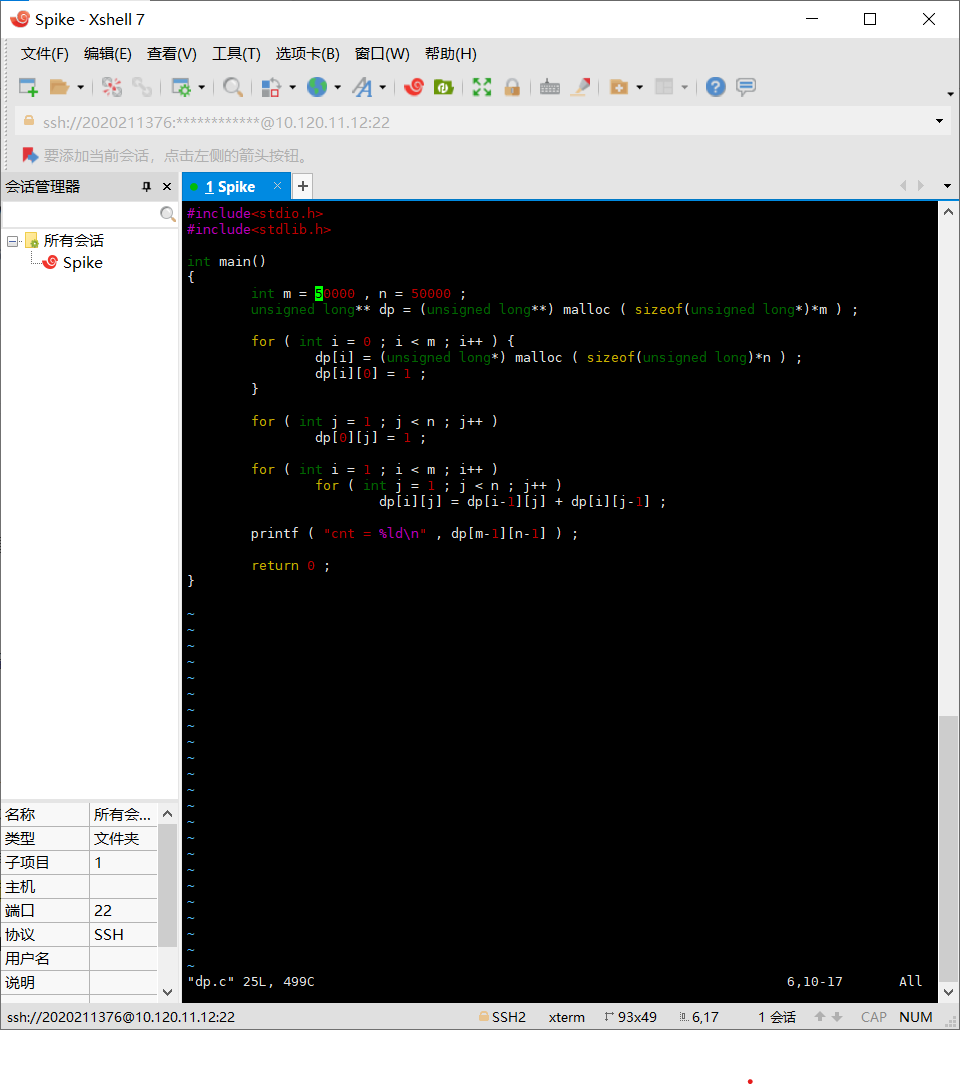
### 1. 高内存需求

问题选择：

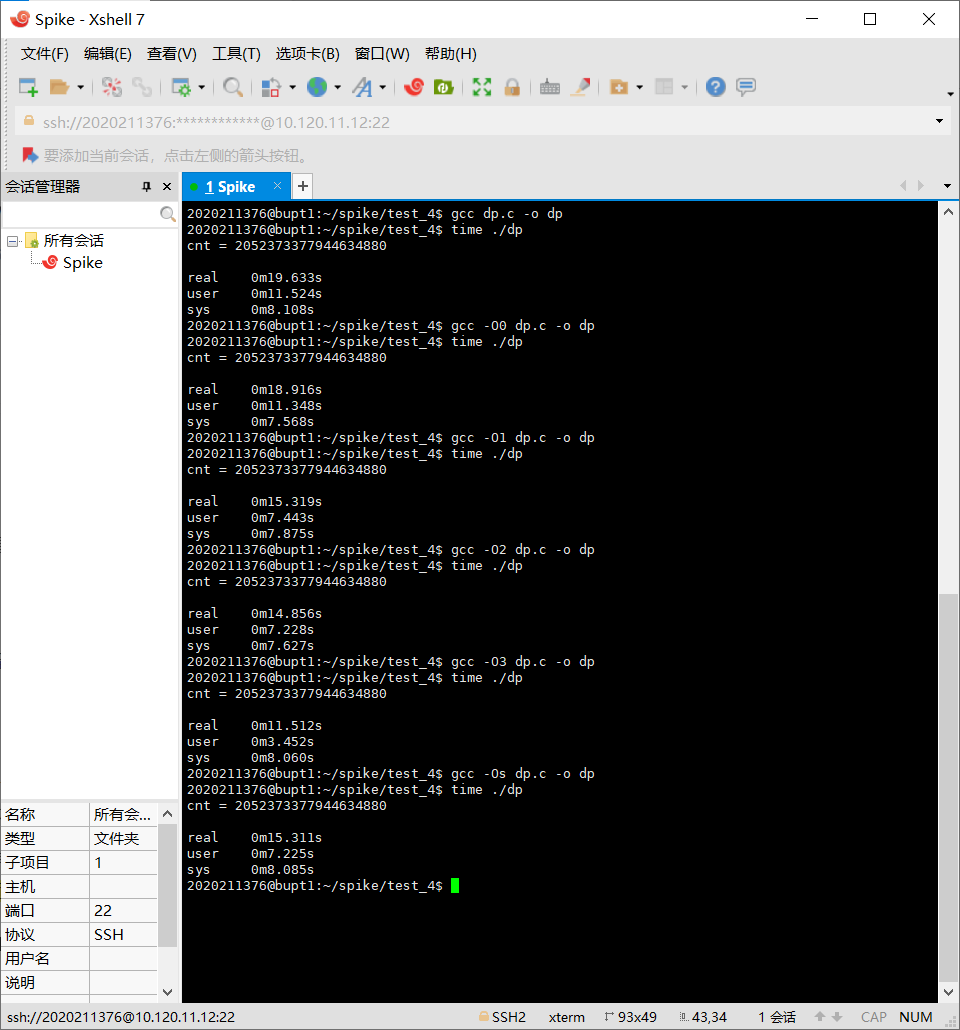
选择高内存的动态规划问题

描述：有100000 \* 100000的方格，每次向右或者向下一格，请问从左上到右下有多少种方案？（数据过大，输出mod (unsigned long)的值）

源代码



优化结果：



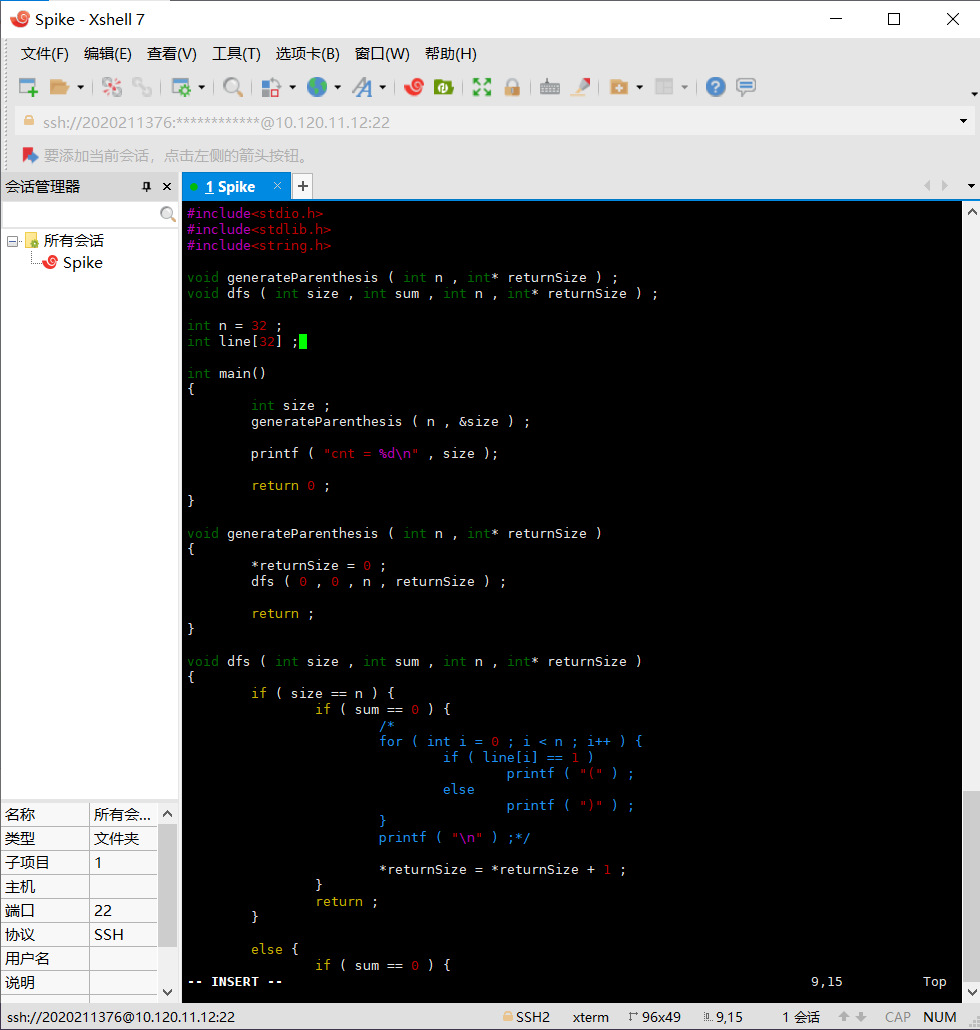
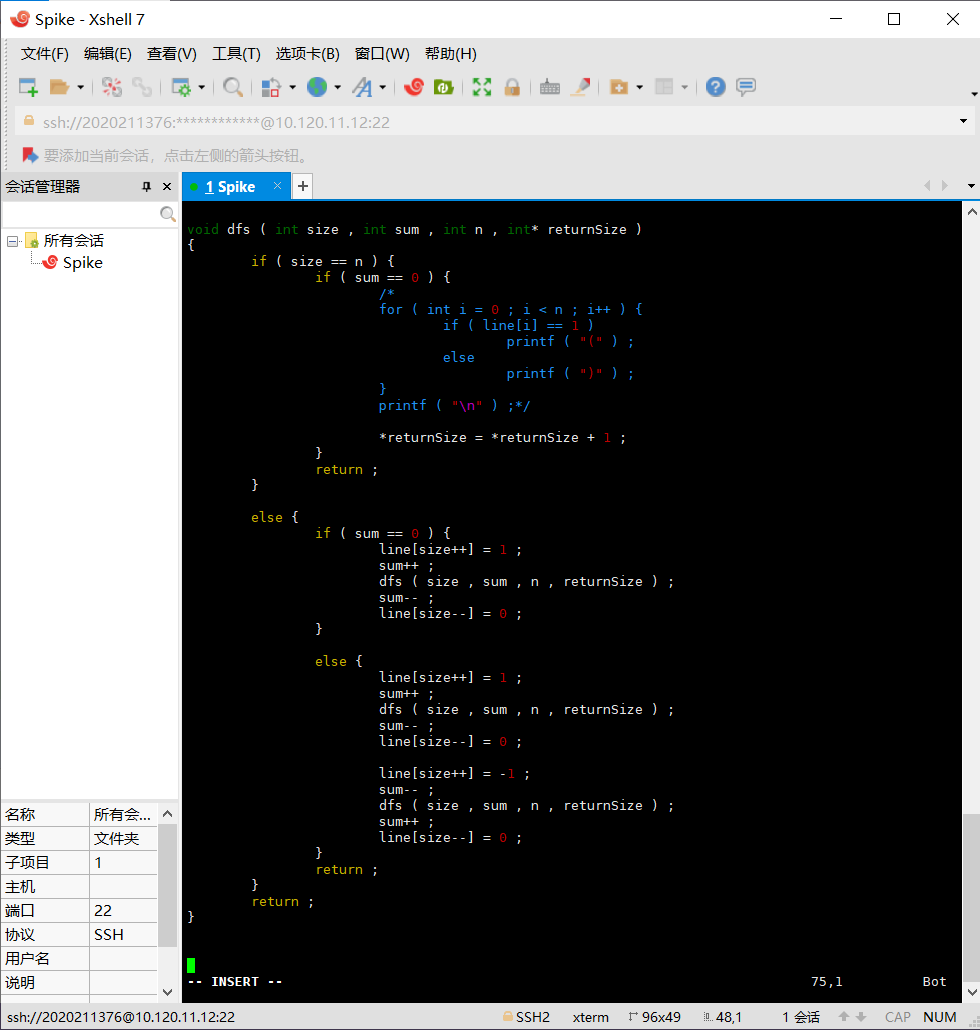
可以推测，O3对高空间的优化更敏感，因为O3的优化效果最好。

### 2. 高时间需求

问题选择：选择指数型问题，并且选择递归来增加时间复杂度

找出长为n=16对括号排列的合法排列种类数。如：n=4: ( ( ) ) , ( ) ( ) 合法 ; ( ) ) ( 不合法

源代码：



1. 递归算法，计( 为1 ， 计 )为-1；

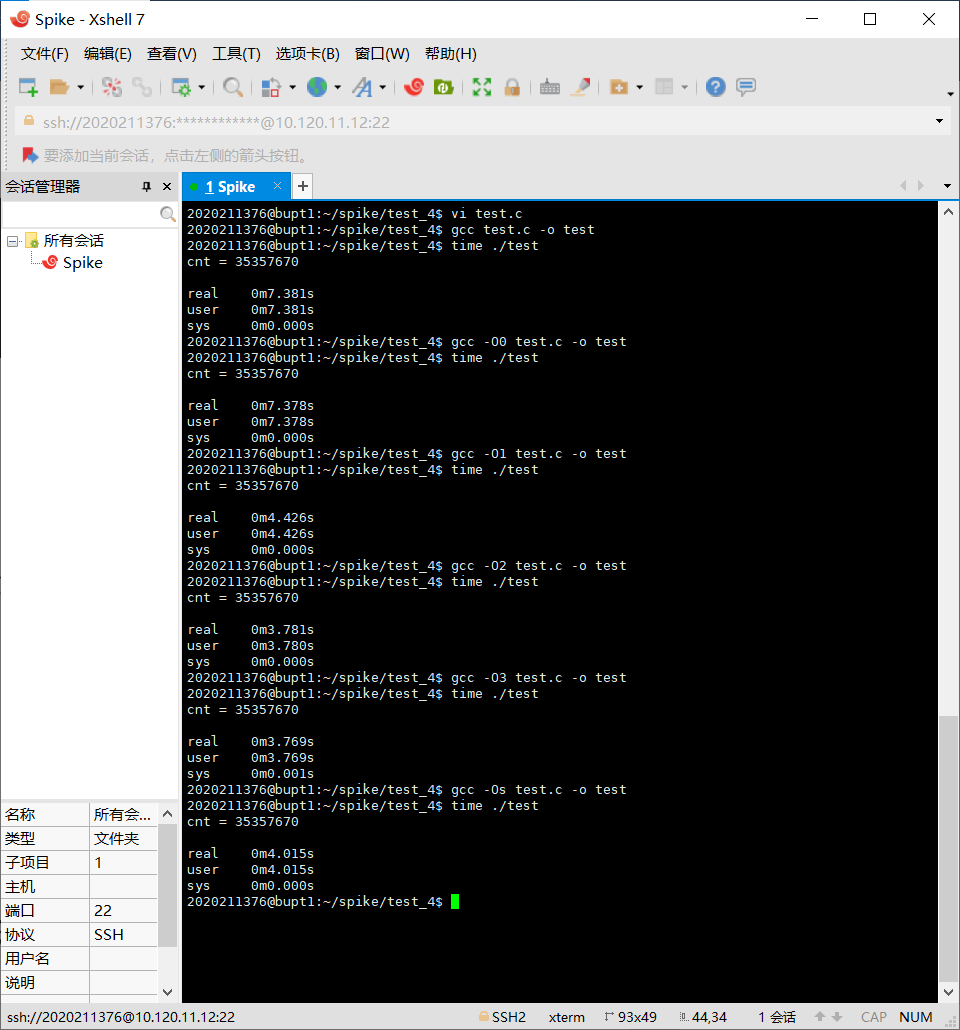
2. line为括号 1，-1的记录数组；

3. size为当前已经填充的括号数量（1，-1代替）；

4. sum为当前一旦小于0则return；

5. 不断地更新参数去逐步递归，到终点时判断是否符合合法条件，符合则计数器加一。

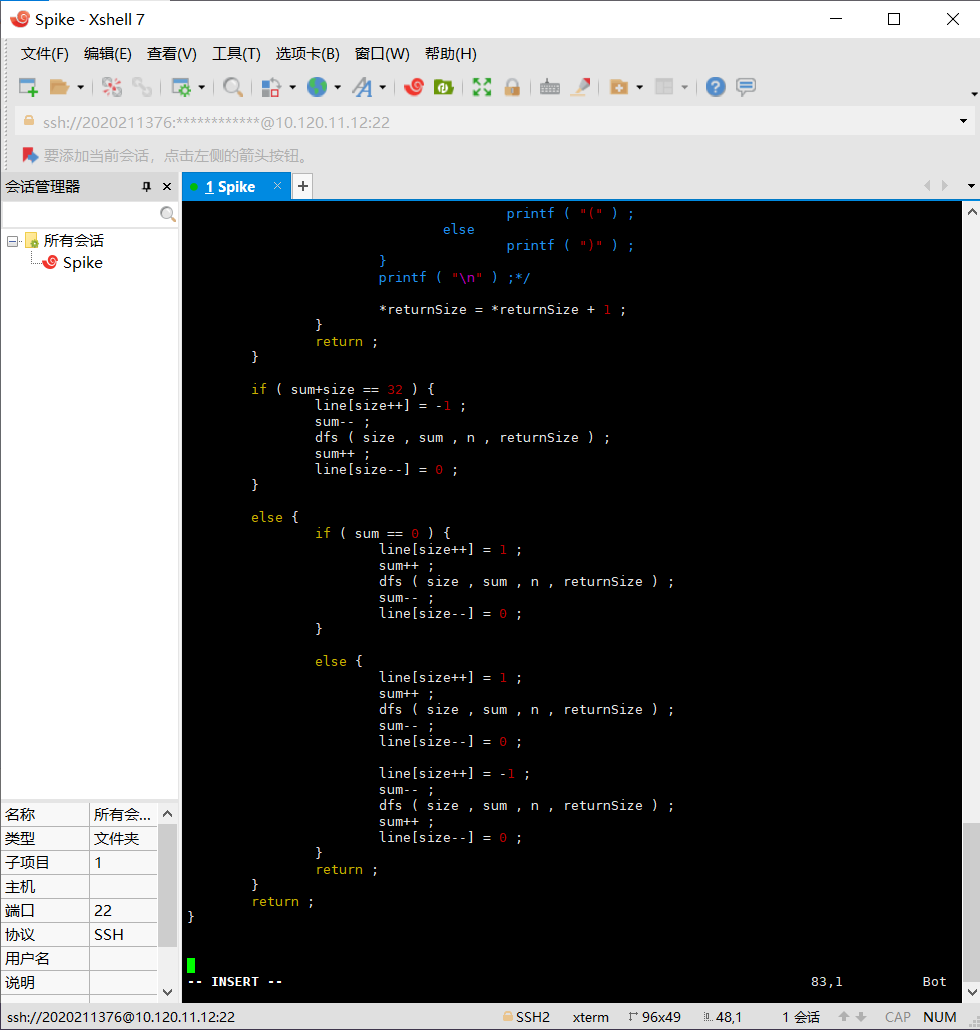
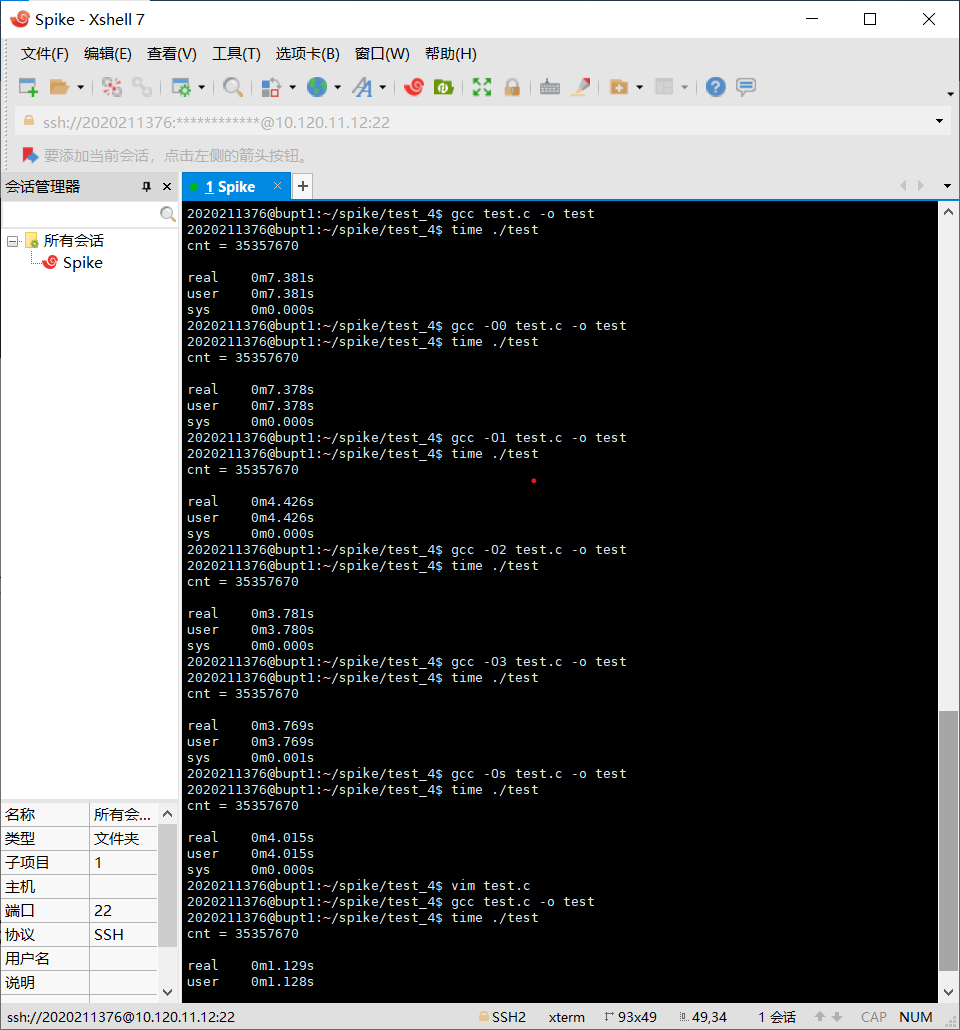
编译优化如下：



初始代码我并没有加限制，导致左括号直接加，到32个填完了才判断是否左括号太多。

所以我优化了我的算法，在加括号的时候加了限制，当左括号数达到一半时只能增加右括号。所以即使是未优化，时间依然比之前的优化快得多：

代码：加了判断左括号数达到32： 优化后时间：(vim进行修改)



显然，快了很多；如果再去掉递归用栈实现，那么我的程序将会更快。

所以，代码优化要自己来，首先是优化算法，再是牺牲一些空间，目的就是减小时间复杂度。

一定不能写垃圾代码。

# **五、实验分析**

工作思路：

1. 先看PPT，了解相关操作；

2. 找一个代码上手，熟悉相关操作；

3. 按照实验步骤来一步步执行，并且思考相关问题和理解；

4. 整理反思，了解linux的操作思想，熟悉操作习惯。

工作内容思路：

将内容紧密地展现，尽量还原指令的使用场景，使得内容不突兀。

效果：

1. 深入了解了实验内容和实验思想，对linux操作系统的相关使用已经有了初步的了解。

2. 主要学习了gdb，gcc，objdump的使用以及在远程连接中管理自己的文件。

3. 初步了解怎么通过反汇编的操作去debug自己的代码。

4. 了解了解放鼠标的操作系统，并熟悉上手。

5. 通过gdb debug过程我也对以前的调试信息有了更深的了解。

# **六、实验总结**

这里我总结一些遇到的问题和体会心得：

1. 上手：因为是完全解放鼠标的操作系统，一开始上手很不适应

2. vim对新手太不友好，根本不会用；习惯性的Ctrl+S使得屏幕直接冻结，不会解锁。

3. 在输入指令时经常对空格把握不好导致指令报错或无法执行

4. 在使用objdump反编译的时候不知道该反编译哪个文件，即不了解编译过程。

5. 在使用gdb调试时一开始很傻，不知道断点能加判断条件导致通过观察寄存器加stepi来执行到相应步骤。

6. 在执行gdb调试时一开始不知道调试出来的是什么，也不知道观察寄存器，也不知道打印地址存的值。

7. 该操作系统对控件的使用非常敏感，尤其是在子函数中进行malloc，能给的空间很小。于是我在主函数进行申请就解决了。

但是，以上难题我都在该实验的完成中成功解决，非常感谢老师能布置这样一个课题。

所以，只有动手实践，才能深入理解。

# **七、诚信声明**

需要填写如下声明，并在底部给出手写签名的电子版。

我还参考了以下资料：我自己收集的笔记，记录成md文件，主要是为了快速上手和了解实验（objdump，gcc优化参数，gdb相关指令使用方法）

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。

（签名）