

哈尔滨工业大学

实验报告

实验（二）

题 目 DataLab 数据表示

专 业 计算机类

学 号 1190501614

班 级 1903006

学 生 cgh

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2021.04.10

计算机科学与技术学院

目 录

第 1 章 实验基本信息	- 4 -
1.1 实验目的	- 4 -
1.2 实验环境与工具	- 4 -
1.2.1 硬件环境	- 4 -
1.2.2 软件环境	- 4 -
1.2.3 开发工具	- 4 -
1.3 实验预习	- 4 -
第 2 章 实验环境建立	- 6 -
2.1 UBUNTU 下 CODEBLOCKS 安装	- 6 -
2.2 64 位 UBUNTU 下 32 位运行环境建立	- 6 -
第 3 章 C 语言的数据类型与存储	- 7 -
3.1 类型本质 (1 分)	- 7 -
3.2 数据的位置-地址 (2 分)	- 7 -
3.3 MAIN 的参数分析 (2 分)	- 10 -
3.4 指针与字符串的区别 (2 分)	- 11 -
第 4 章 深入分析 UTF-8 编码	- 12 -
4.1 提交 UTF8LEN.C 子程序	- 12 -
4.2 C 语言的 STRCMP 函数分析	- 12 -
4.3 讨论: 按照姓氏笔画排序的方法实现	- 12 -
第 5 章 数据变换与输入输出	- 13 -
5.1 提交 CS_ATOL.C	- 13 -
5.2 提交 CS_ATOF.C	- 13 -
5.3 提交 CS_ITOA.C	- 13 -
5.4 提交 CS_FTOA.C	- 13 -
5.5 讨论分析 OS 的函数对输入输出的数据有类型要求吗	- 13 -
第 6 章 整数表示与运算	- 14 -
6.1 提交 FIB_DG.C	- 14 -
6.2 提交 FIB_LOOP.C	- 14 -
6.3 FIB 溢出验证	- 14 -
6.4 除以 0 验证:	- 14 -
6.5 万年虫验证	- 15 -
6.6 2038 虫验证	- 16 -
第 7 章 浮点数据的表示与运算	- 18 -

7.1 手动 FLOAT 编码:	- 18 -
7.2 特殊 FLOAT 数据的处理.....	- 18 -
7.3 验证浮点运算的溢出	- 19 -
7.4 类型转换的坑	- 19 -
7.5 讨论 1: 有多少个 INT 可以用 FLOAT 精确表示	- 20 -
7.6 讨论 2: 怎么验证 FLOAT 采用的向偶数舍入呢.....	- 20 -
7.7 讨论 3: FLOAT 能精确表示几个 1 元内的钱呢	- 20 -
7.8 FLOAT 的微观与宏观世界	- 20 -
7.9 讨论: 浮点数的比较方法.....	- 21 -
第 8 章 舍尾平衡的讨论.....	- 22 -
8.1 描述可能出现的问题.....	- 22 -
8.2 给出完美的解决方案.....	- 22 -
第 9 章 总结	- 23 -
9.1 请总结本次实验的收获.....	- 23 -
9.2 请给出对本次实验内容的建议	- 23 -
参考文献.....	- 24 -

第 1 章 实验基本信息

1.1 实验目的

熟练掌握计算机系统的数据表示与数据运算

通过 C 程序深入理解计算机运算器的底层实现与优化

掌握 VS/CB/GCC 等工具的使用技巧与注意事项

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU; 2GHz; 2G RAM; 256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上; VirtualBox/Vmware 11 以上; Ubuntu 16.04 LTS 64 位/
优麒麟 64 位;

1.2.3 开发工具

Visual Studio 2010 64 位以上; CodeBlocks; vi/vim/gpedit+gcc

1.3 实验预习

上实验课前, 必须认真预习实验指导书 (PPT 或 PDF)

了解实验的目的、实验环境与软硬件工具、实验操作步骤, 复习与实验有关的理论知识。

采用 `sizeof` 在 Windows 的 VS/CB 以及 Linux 的 CB/GCC 下获得 C 语言每一类型在 32/64 位模式下的空间大小

编写 C 程序, 计算斐波那契数列在 `int/long/unsigned int/unsigned long` 类型时, `n` 为多少时会出错

写出 **float/double** 类型最小的正数、最大的正数（非无穷）

按步骤写出 **float** 数-1.1 在内存从低到高地址的字节值-16 进制

按照阶码区域写出 **float** 的最大密度区域范围及其密度，最小密度区域及其密度（区域长度/表示的浮点个数）

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 安装

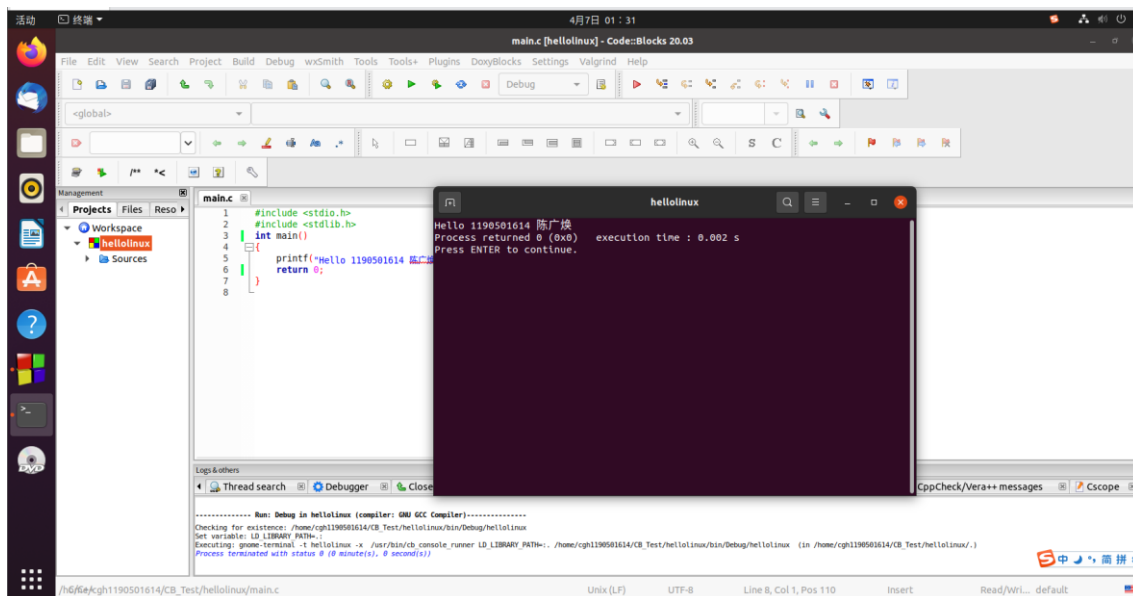
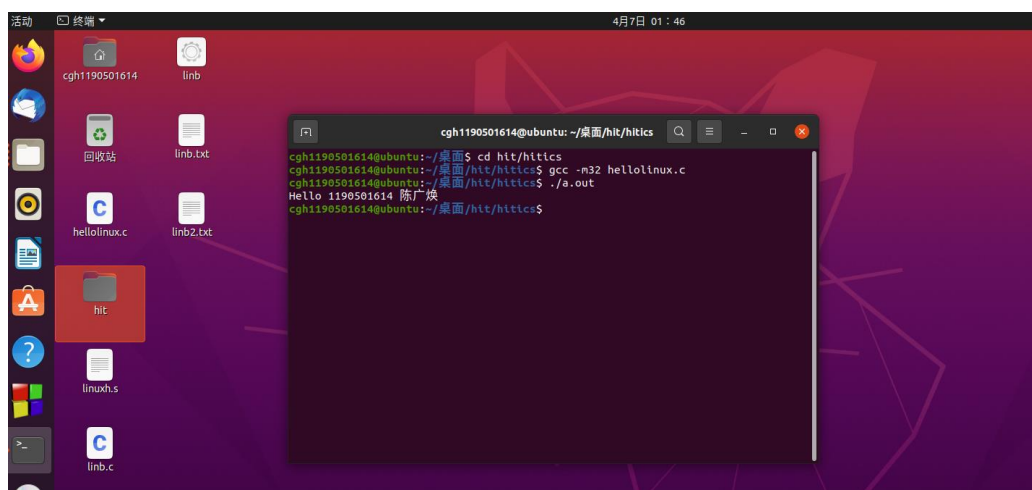


图 2-1 Ubuntu 下 CodeBlocks 截图

2.2 64 位 Ubuntu 下 32 位运行环境建立



第 3 章 C 语言的数据类型与存储

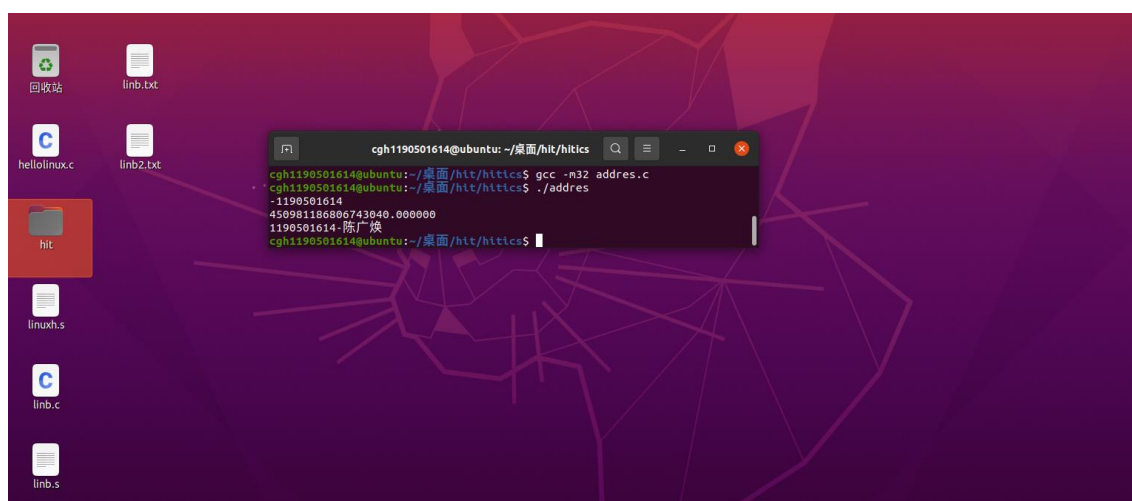
3.1 类型本质

	Win/VS/x86	Win/VS/x64	Win/CB/32	Win/CB/64	Linux/CB/32	Linux/CB/64
char	1	1	1	1	1	1
short	2	2	2	2	2	2
int	4	4	4	4	4	4
long	4	4	4	8	4	8
long long	8	8	8	8	8	8
float	4	4	4	4	4	4
double	8	8	8	8	8	8
long double	8	8	12	16	12	16
指针	4	8	4	8	4	8

C 编译器对 sizeof 的实现方式：在编译阶段就计算出结果，在运行时是个常量。因为编译阶段可以确定数据类型，根据数据类型换算数据的长度

3.2 数据的位置-地址

打印 x、y、z 输出的值：截图 1



反汇编查看 x、y、z 的地址，每字节的内容：截图 2，标注说明 GDB 调试结果

```

Breakpoint 1, main () at address.c:9
9      printf("%d\n",x);
(gdb) p &x
$1 = (const int *) 0x555555556004 <x>
(gdb) p x
$2 = -1190501614
(gdb) p &y
$3 = (const float *) 0x7fffffffdddec
(gdb) p y
$4 = 4.50981187e+17
(gdb) p &z
$5 = (char (*)[21]) 0x555555558010 <z>
(gdb) p z
$6 = "1190501614-陈广焕"
(gdb) x/4xb &x
0x555555556004 <x>:  0x12  0x63  0x0a  0xb9
(gdb) x/4xb &y
0x7fffffffdddec: 0xa0  0x46  0xc8  0x5c
(gdb) x/4xb &z
0x555555558010 <z.2317>:  0x31  0x31  0x39  0x30
(gdb)

```

反汇编结果

```

00004008 <z.1910>:
4008: 31 31 39 30 35 30 31 36 31 34 2d e9 99 88 e5 b9 1190501614-.....
4018: bf e7 84 95 00 00 00 00 00 00 00 00 00 00 00 .....
cgh1190501614@ubuntu:~/桌面/hit/hitics$ objdump -j .rodata -S a.out

a.out: 文件格式 elf32-i386

Disassembly of section .rodata:

00002000 <_fp_hw>:
2000: 03 00 00 00 ....

00002004 <_IO_stdin_used>:
2004: 01 00 02 00 ....

00002008 <x>:
#include<stdio.h>

const int x = -1190501614;
2008: 12 63 0a b9 25 64 0a 00 25 66 0a 00 a0 46 c8 5c .c..%d..%f...F.\
cgh1190501614@ubuntu:~/桌面/hit/hitics$

```

```

cgh1190501614@ubuntu: ~/桌面/hit/hitics
000011e9 <__x86.get_pc_thunk.dx>:
11e9: 8b 14 24      mov    (%esp),%edx
11ec: c3           ret

000011ed <main>:
11ed: f3 0f 1e fb   endbr32
11f1: 8d 4c 24 04   lea    0x4(%esp),%ecx
11f5: 83 e4 f0     and    $0xfffffff0,%esp
11f8: ff 71 fc     pushl -0x4(%ecx)
11fb: 55           push   %ebp
11fc: 89 e5       mov    %esp,%ebp
11fe: 53           push   %ebx
11ff: 51           push   %ecx
1200: 83 ec 10     sub    $0x10,%esp
1203: e8 e8 fe ff ff call   10f0 <__x86.get_pc_thunk.bx>
1208: 81 c3 cc 2d 00 00 add    $0x2dcc,%ebx
120e: d9 83 40 e0 ff ff flds   -0x1fc0(%ebx)
1214: d9 5d f4     fstps  -0xc(%ebp)
1217: b8 12 63 0a b9 mov    $0xb90a0312,%eax
121c: 83 ec 08     sub    $0x8,%esp
121f: 50           push   %eax
1220: 8d 83 38 e0 ff ff lea    -0x1fc8(%ebx),%eax
1226: 50           push   %eax
1227: e8 54 fe ff ff call   1080 <printf@plt>
122c: 83 c4 10     add    $0x10,%esp
122f: d9 45 f4     flds   -0xc(%ebp)
1232: 83 ec 04     sub    $0x4,%esp
1235: 8d 64 24 f8   lea    -0x8(%esp),%esp

```



```

objdump: x: 无此文件
cgh1190501614@ubuntu: ~/桌面/hit/hitics$ objdump -j .data -S a.out

a.out:          文件格式 elf32-i386

Disassembly of section .data:

00004000 <__data_start>:
   4000:          00 00                                add    %al,(%eax)
   ...

00004004 <__dso_handle>:
   4004:          04 40 00 00                        .@..

00004008 <z.1910>:
   4008:          31 31 39 30 35 30 31 36 31 34 2d e9 99 88 e5 b9    1190501614-.....
   4018:          bf e7 84 95 00 00 00 00 00 00 00 00 00 00 00    .....

```

反汇编查看 x、y、z 在代码段的表示形式。截图 3，标注说明

```

cgh1190501614@ubuntu: ~/桌面/hit/hitics

12f4:      e8 f7 fd ff ff      call    10f0 <__x86.get_pc_thunk.bx>
12f9:      81 c3 db 2c 00 00    add     $0x2cdb,%ebx
12ff:      83 c4 08             add     $0x8,%esp
1302:      5b                  pop     %ebx
1303:      c3                  ret

Disassembly of section .rodata:

00002000 <_fp_hw>:
   2000:          03 00                                add     (%eax),%eax
   ...

00002004 <_IO_stdin_used>:
   2004:          01 00                                add     %eax,(%eax)
   2006:          02 00                                add     (%eax),%al

00002008 <x>:
   2008:          12 63 0a             adc     0xa(%ebx),%ah
   200b:          b9 25 64 0a 00        mov     $0xa6425,%ecx
   2010:          25 66 0a 00 a0        and     $0xa000a66,%eax
   2015:          46                     inc     %esi
   2016:          c8                     .byte 0xc8
   2017:          5c                     pop     %esp

Disassembly of section .eh_frame_hdr:

```

```

Disassembly of section .rodata:

00002000 <_fp_hw>:
   2000:          03 00                                add     (%eax),%eax
   ...

00002004 <_IO_stdin_used>:
   2004:          01 00                                add     %eax,(%eax)
   2006:          02 00                                add     (%eax),%al

```

```

Disassembly of section .data:

00004000 <__data_start>:
  4000:    00 00                add    %al,(%eax)
  ...

00004004 <__dso_handle>:
  4004:    04 40                add    $0x40,%al
  ...

00004008 <z.1910>:
  4008:    31 31                xor    %esi,(%ecx)
  400a:    39 30                cmp    %esi,(%eax)
  400c:    35 30 31 36 31       xor    $0x31363130,%eax
  4011:    34 2d                xor    $0x2d,%al
  4013:    e9 99 88 e5 b9       jmp    b9e5c8b1 <__TMC_END__+0xb9e58889>
  4018:    bf e7 84 95 00       mov    $0x9584e7,%edi
  ...

Disassembly of section .bss:

00004026 <completed.7622>:
  ...

Disassembly of section .comment:

```

68

x 与 y 在_汇编_阶段转换成补码与 ieee754 编码。

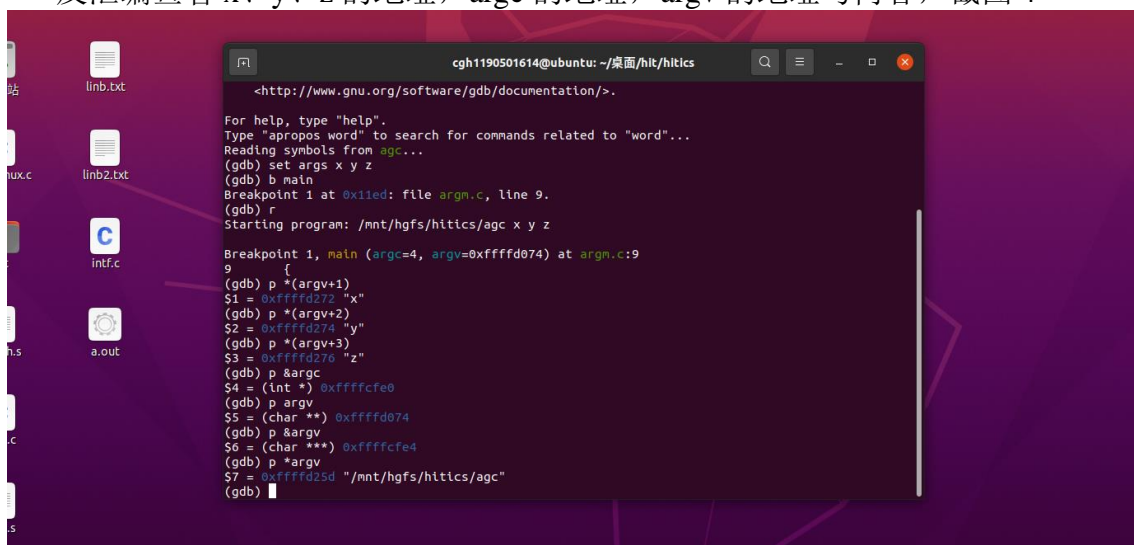
数值型常量与变量在存储空间上的区别是：__const 常量存储在代码段，静态变量和全局变量存储在数据段，局部变量存储在堆栈段 ____

字符串常量与变量在存储空间上的区别是：_字符串常量存储在静态内存区,字符串变量存储在堆栈区或者静态区（全局）____

常量表达式在计算机中处理方法是：__在编译阶段转换成常量来存储____

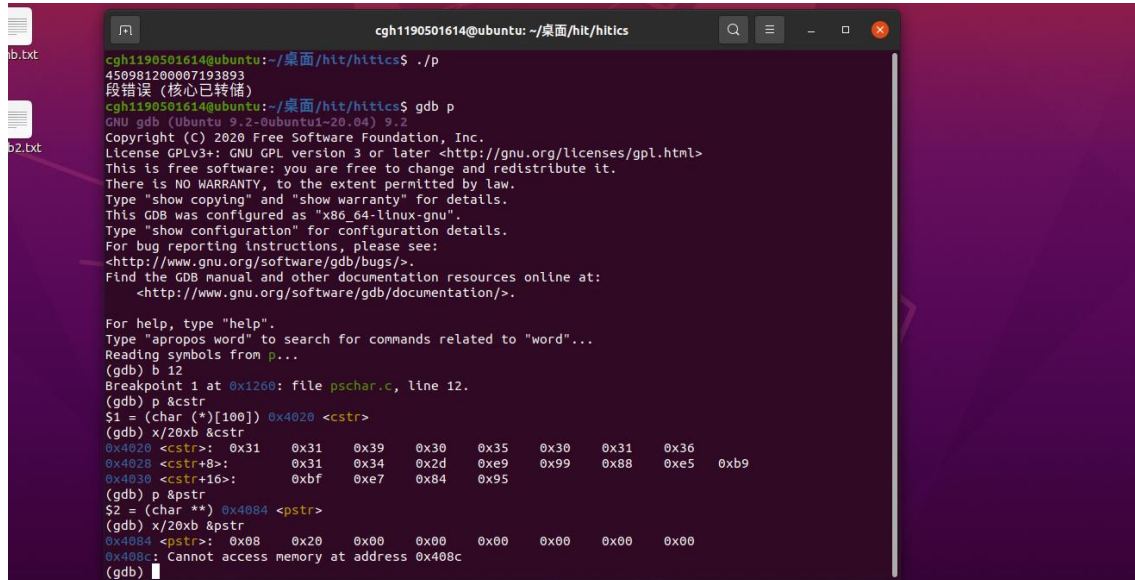
3.3 main 的参数分析

反汇编查看 x、y、z 的地址，argc 的地址，argv 的地址与内容，截图 4



3.4 指针与字符串的区别

cstr 的地址与内容截图，pstr 的内容与截图，截图 5



```
cgh1190501614@ubuntu: ~/桌面/hit/hitics$ ./p
450981200007193893
段错误 (核心已转储)
cgh1190501614@ubuntu:~/桌面/hit/hitics$ gdb p
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
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 online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from p...
(gdb) b 12
Breakpoint 1 at 0x1260: file pschar.c, line 12.
(gdb) p &cstr
$1 = (char *) [100] 0x4020 <cstr>
(gdb) x/20xb &cstr
0x4020 <cstr>: 0x31 0x31 0x39 0x30 0x35 0x30 0x31 0x36
0x4028 <cstr+8>: 0x31 0x34 0x2d 0xe9 0x99 0x88 0xe5 0xb9
0x4030 <cstr+16>: 0xbf 0xe7 0x84 0x95
(gdb) p &pstr
$2 = (char **) 0x4084 <pstr>
(gdb) x/20xb &pstr
0x4084 <pstr>: 0x08 0x20 0x00 0x00 0x00 0x00 0x00 0x00
0x408c: Cannot access memory at address 0x408c
(gdb)
```

pstr 修改内容会出现什么问题___无法修改，显示段错误（核心已转储）

第 4 章 深入分析 UTF-8 编码

4.1 提交 utf8len.c 子程序

见附件

4.2 C 语言的 strcmp 函数分析

每个汉字都有其对应的 unicode 码，strcmp 函数按照汉字对应的 unicode 编码大小对汉字进行排序。每个汉字所占字节可能不同，但 strcmp 函数会按照汉字的 Unicode 编码逐个字节比较，直到遇到结束字符编码。

4.3 讨论：按照姓氏笔画排序的方法实现

规则一：姓按字的画数多少排列。同笔画的姓字按起笔一、丨、丿、丶、乚的顺序排列，画数和笔形相同的字，按字形结构，先左右形字，再上下形字，后整体字

规则二：姓字相同的，单字名排在多字名之前；多字名依次看名的第一、第二字……先看画数，后看起笔顺序，再看笔形。

规则三：复姓字也按第一个字笔画画数多少排列，笔画相同的按笔形顺序排列

按照规则一和规则三，对所有的汉字进行排序，然后将得到的排序结果与汉字的 Unicode 编码一一对应，首先比较姓氏，通过读取其 Unicode 编码找到其对应的序号，当序号相同（姓氏相同）时，比较名字，根据规则二进行排序。

第 5 章 数据变换与输入输出

5.1 提交 `cs_atoi.c`

5.2 提交 `cs_atof.c`

5.3 提交 `cs_itoa.c`

5.4 提交 `cs_ftoa.c`

5.5 讨论分析 OS 的函数对输入输出的数据有类型要求吗

论述如下：

应用程序是通过分别调用 `read` 和 `write` 函数来执行输入和输出的。

```
ssize_t read (int fd,void *buf,size_t n)
```

```
ssize_t write (int fd,void *buf,size_t n)
```

`read` 函数从描述符为 `fd` 的当前文件位置复制最多 `n` 个字节到内存位置 `buf`，返回值为 -1 表示一个错误，返回值为 0 表示 EOF，否则，返回值为实际传送的字节数量

`Write` 函数从描述符为 `buf` 的当前文件位置复制最多 `n` 个字节到描述符 `buf` 的当前位置。

第 6 章 整数表示与运算

6.1 提交 fib_dg.c

见附件

6.2 提交 fib_loop.c

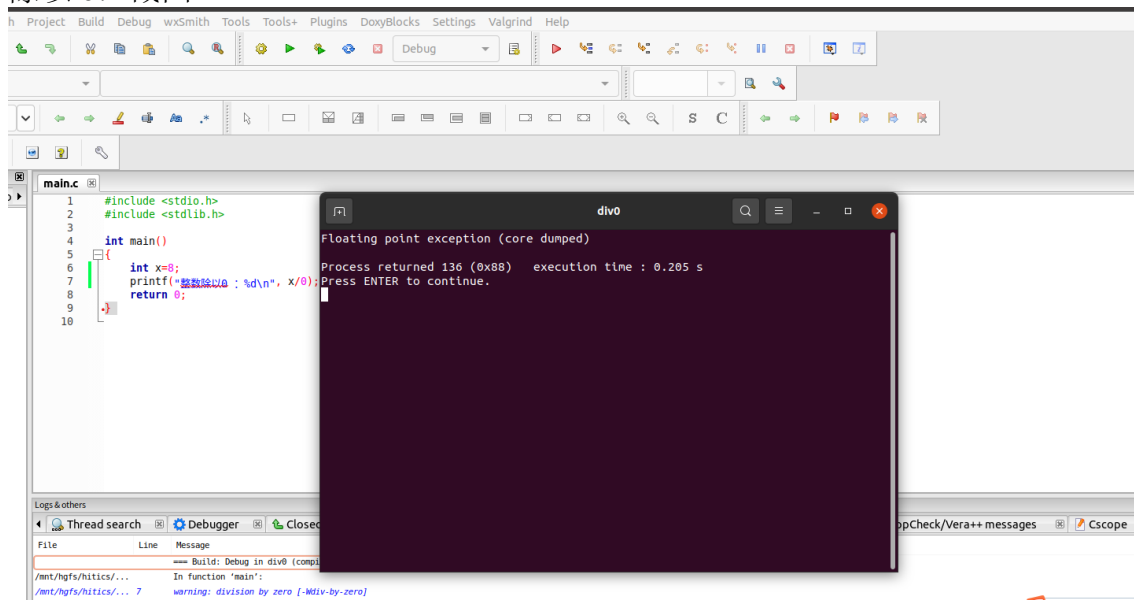
见附件

6.3 fib 溢出验证

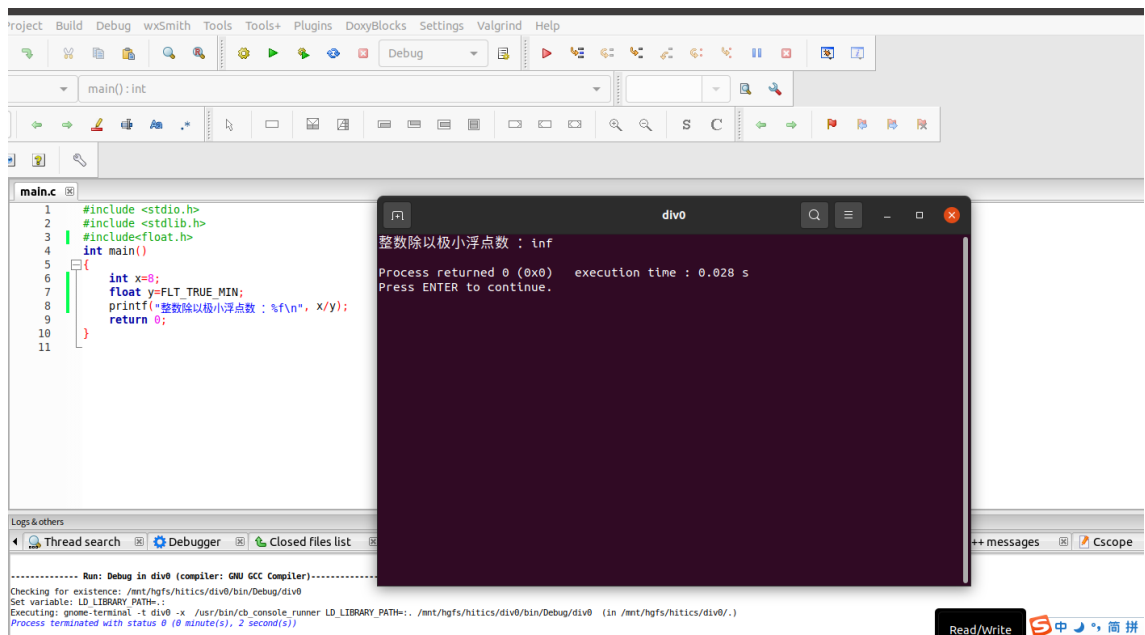
int 时从 n=__47__时溢出, long 时 n=__47__时溢出。
unsigned int 时从 n=__48__时溢出, unsigned long 时 n=__48__时溢出。

6.4 除以 0 验证:

除以 0: 截图 1

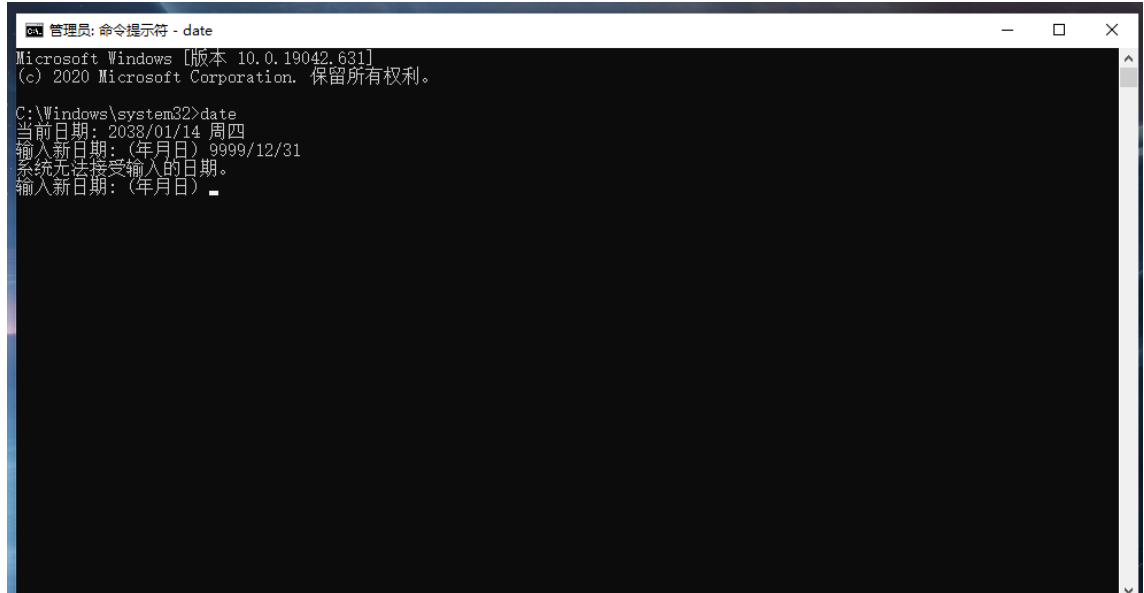


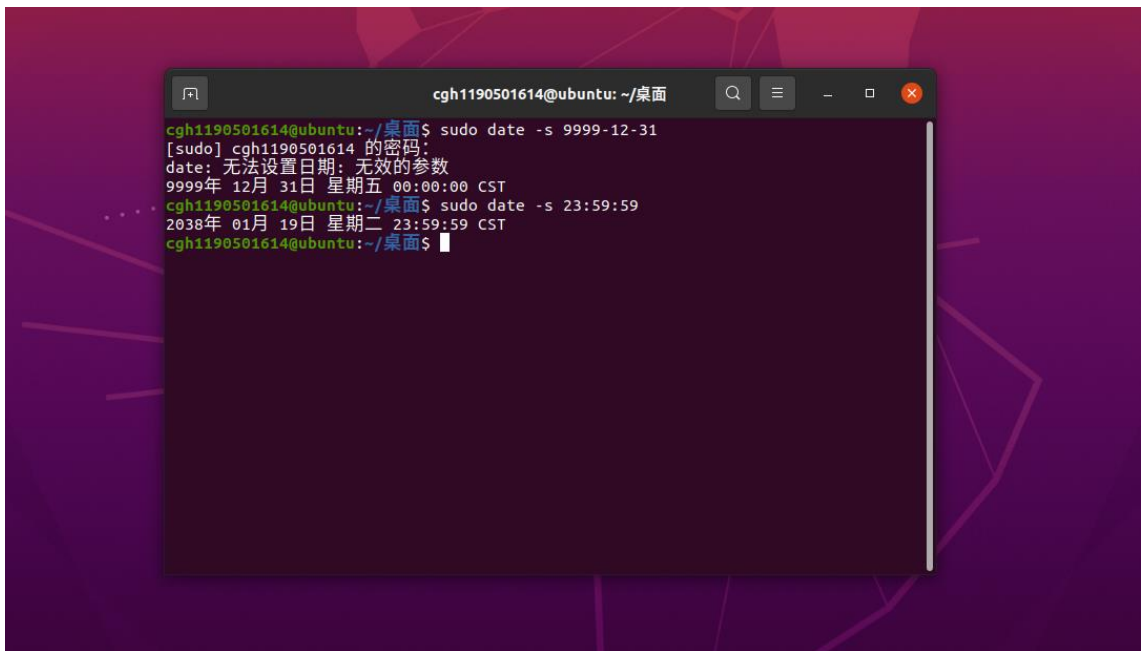
除以极小浮点数, 截图:



6.5 万年虫验证

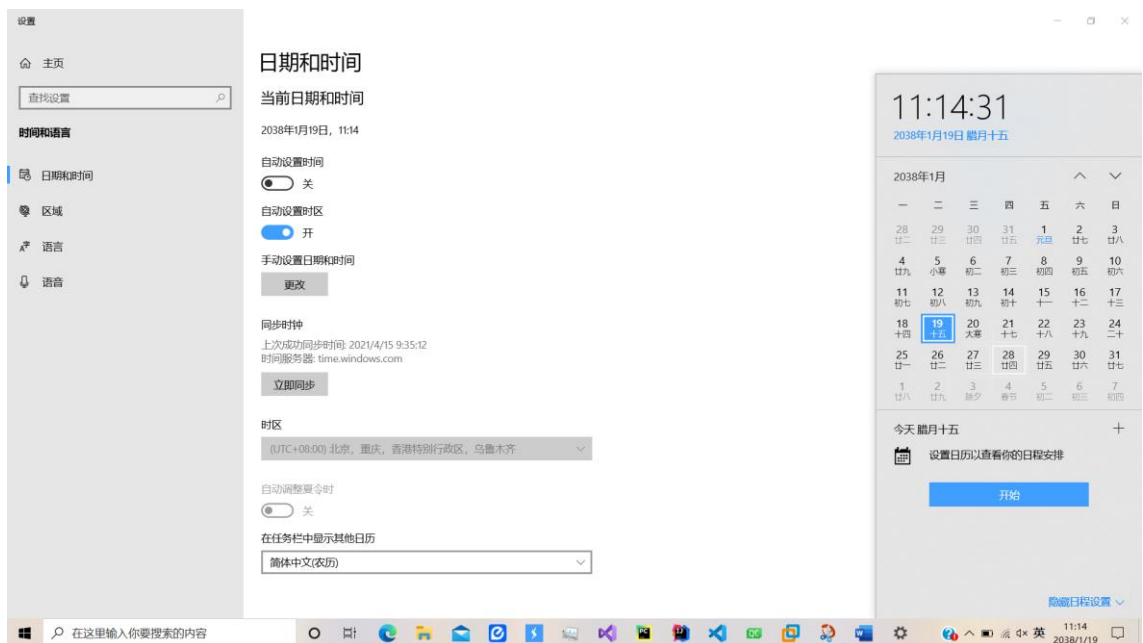
你的机器到 9999 年 12 月 31 日 23:59:59 后,时钟怎么显示的? Windows/Linux 下分别截图:



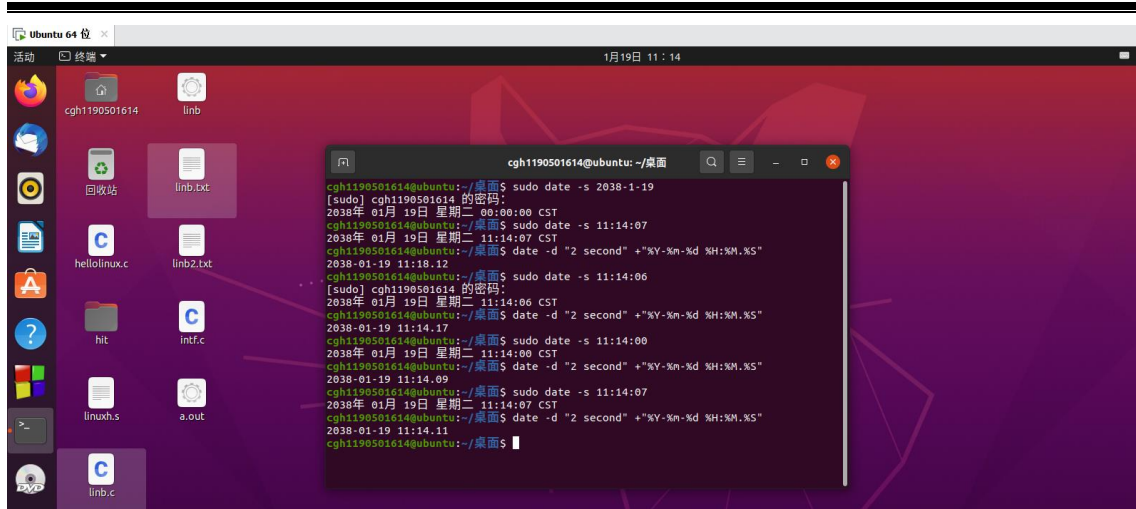


6.6 2038 虫验证

2038 年 1 月 19 日中午 11:14:07 后你的计算机时间是多少，Windows/Linux 下分别截图



计算机系统实验报告



第 7 章 浮点数据的表示与运算

7.1 手动 float 编码：

按步骤写出 float 数-10.1 在内存从低到高地址的字节值（16 进制）。

符号位：1

(1) 整数二进制：1010

(2) 小数部分二进制：0.00011001100110011001100110011

(3) 科学记数法为：-1.0100 001 1001 1001 1001 1001*2⁻³

(4) 指数：3，3+127=130，因此阶码为 1000 0100

(5) 小数部分采用乘 2 取整法得：

因此：符号位 指数部分 尾数部分

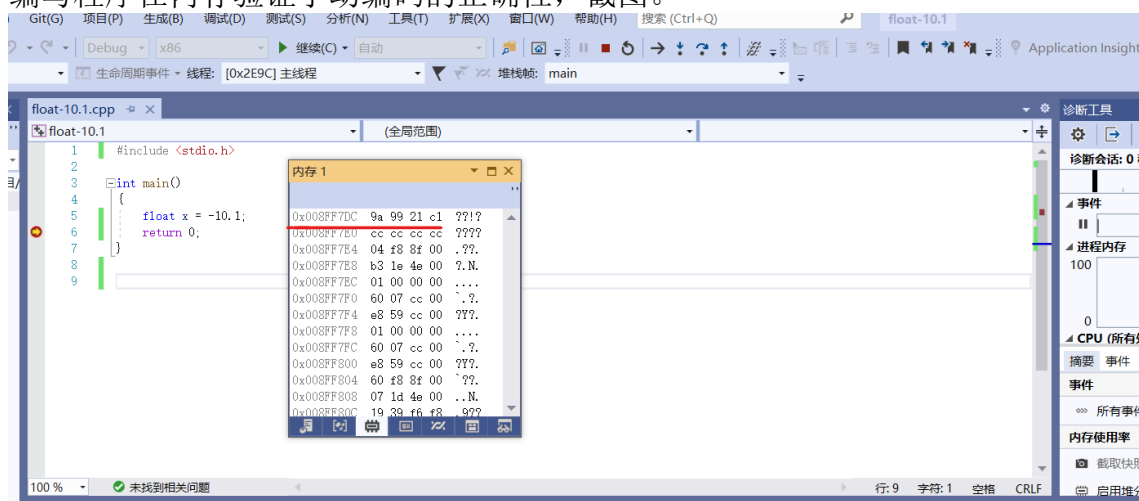
1 1000 0100 0100001 1001 1001 1001 1001

因此从低到高的二进制为：1100 0010 0010 0001 1001 1001 1001 1001

对应的 16 进制为：c121999a

因此低到高字节排序为：9a 99 21 c

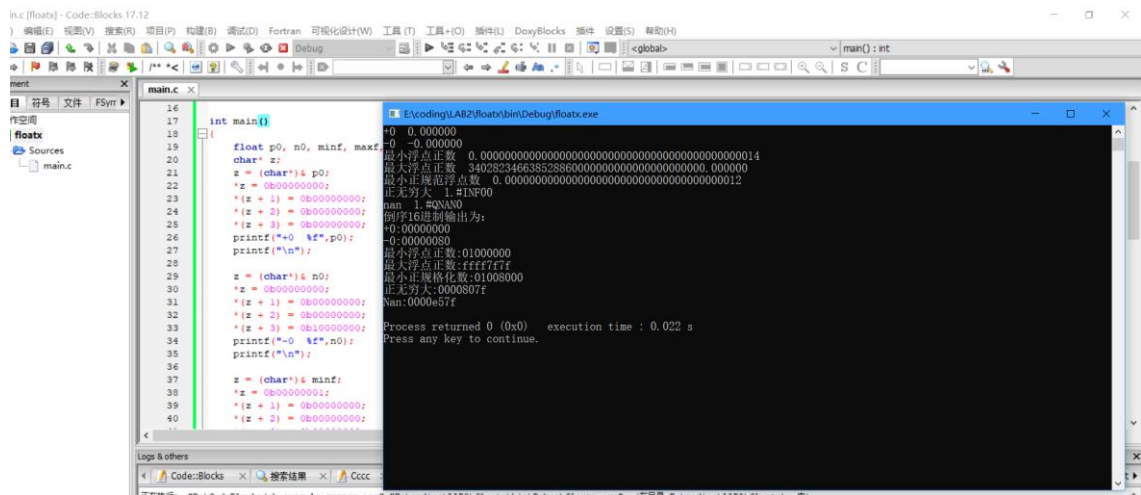
编写程序在内存验证手动编码的正确性，截图。



7.2 特殊 float 数据的处理

提交子程序 floatx.c，要求：

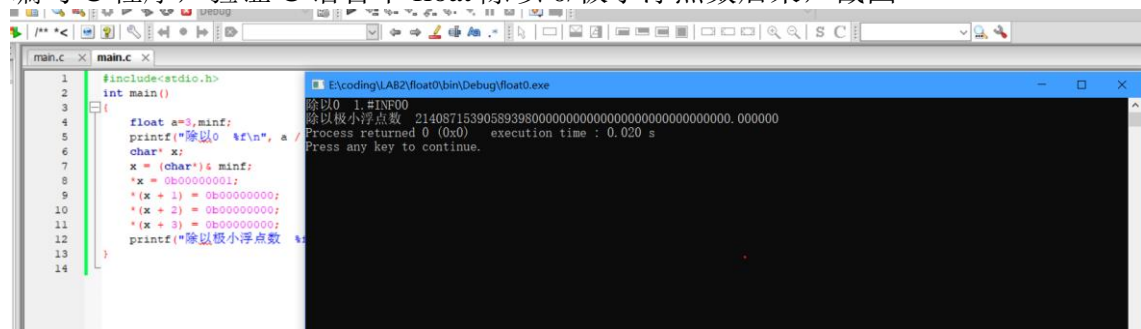
构造多 float 变量，分别存储+0.0，最小浮点正数，最大浮点正数、最小正的规格化浮点数、正无穷大、Nan,并打印最可能的精确结果输出（十进制/16 进制）。截图。



7.3 验证浮点运算的溢出

提交子程序 float0.c

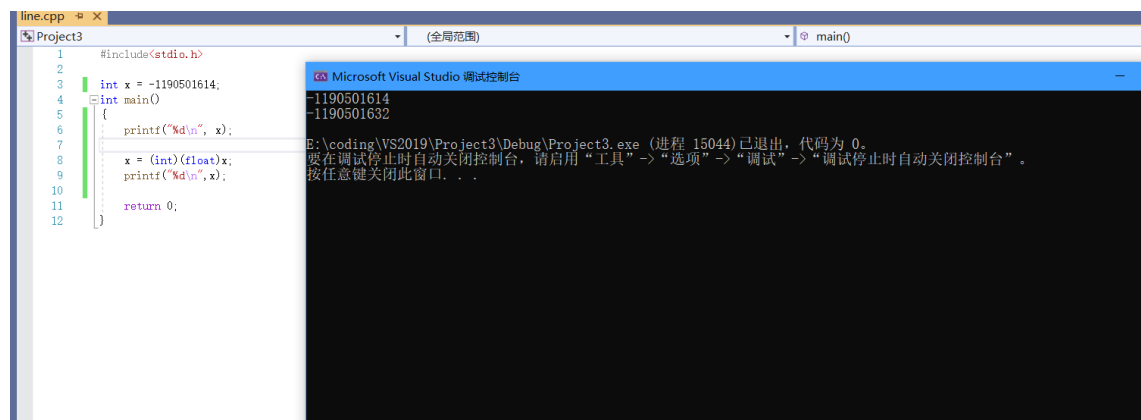
编写 C 程序，验证 C 语言中 float 除以 0/极小浮点数后果，截图



7.4 类型转换的坑

实验指导 PPT 第 5 步骤的 x 变量，执行 `x=(int)(float)x` 后结果为多少？

原 `x=-1190501614`，现 `x=-1190501632`



7.9 讨论：浮点数的比较方法

从键盘输入或运算后得到的任意两个浮点数，论述其比较方法以及理由。

由于计算机存储浮点数的特性，两个浮点数的比较不能直接用`==`运算符比较，一般有效的方法是将两个浮点数做差，如何这个差值的绝对值小于给定的精度，则认为是相等的，否则就是不相等的。

如果其中有一个是 `NAN`，一般认为是不可比较的，如果两个都是 `NAN`，有些编译器会认为是相等的。正 0 和负 0 是相等的。对于更高精度的比较，可根据符号位，阶码和尾数进行比较。

第 8 章 舍尾平衡的讨论

8.1 描述可能出现的问题

舍位处理往往会采取四舍五入计算，这时就会产生误差，而如果报表中有这些数据的合计数值，那么舍位时产生的误差就会积累，有可能导致舍位过的数据与其合计值无法匹配。从而造成一定的损失。

8.2 给出完美的解决方案

- 如果在数据统计时，每个数据只用于一次合计，那么在处理舍位平衡时，只需要根据合计值的误差，调整使用的各项数据就可以了。

首先，将平衡差整理到第一个数据中；

将平衡差按照“最小调整值”，对绝对值比较大的数据进行分担调整。所谓最小调整值，就是舍位后最小精度的单位值，例如在取整时，最小精度就是个位，最小调整值就是 1 或者-1。如果舍位后合计值变小，则需要将数据调大，那么最小调整值就是 1；如果舍位后合计值变大，则需要将数据调小，最小调整值就是-1。而调整只针对绝对值比较大的数据，这样它们的相对偏差就会比较小。

为了提高效率，实际执行过程中，将平衡差按照最小调整值，由不为 0 的数据依次分担。

- 如果数据在行向和列向两个方向同时需要计算合计值，同时还需要计算所有数据的总计值。

首先，横向与纵向的非合计平衡差符号相同，只需要调整交叉点处的数据，根据平衡差符号加减最小调整值即可，

同向的 2 个非合计平衡差符号相反，任选一行平衡差为 0 的数据，将这两列的数分别根据按平衡差的符号加减最小调整值。

某个合计平衡差与另一方向的非合计平衡差符号相反，只需要调整交叉点处的合计数据，根据合计平衡差的符号加减最小调整值。

某个合计平衡差与同方向的非合计平衡差符号相同，可以任选 1 行平衡差为 0 的数据，同时调整这 2 列的数据。

两个方向合计平衡差的符号相同，可以任选一个非合计值，根据合计平衡差的符号加减最小调整值，同样调整这个数据的横向和纵向合计值。

第 9 章 总结

9.1 请总结本次实验的收获

本次实验收获颇丰，学会如何通过 `gdb` 查找变量的地址和内存中的字节表示，对浮点数的认识又更深了一步，同时对反汇编文件的整体结构有了一定了解，知道了全局变量和静态变量的存储位置。对命令行的使用也更加娴熟，对计算机系统的思考更深一层。

9.2 请给出对本次实验内容的建议

Ppt 中的有些语句表述得模棱两可，容易产生错误理解，希望老师可以写得更详细一些，比如探索数据存储位置那个实验。

实验报告存在错误，ppt 上面定义浮点数密度为浮点数/区域长度，而实验报告上面却是恰好相反。

有些知识点可以写得详细一些，给些提示。

参考文献