

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 1190201816

班 级 1903012

学 生 樊红雨

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2021.3.31

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	- 7 -
第 3 章 C 语言的数据类型与存储	- 8 -
3.1 类型本质 (1 分)	- 8 -
3.2 数据的位置-地址 (2 分)	- 8 -
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	- 13 -
4.1 提交 UTF8LEN.C 子程序	- 13 -
4.2 C 语言的 STRCMP 函数分析	- 13 -
4.3 讨论: 按照姓氏笔画排序的方法实现	- 13 -
第 5 章 数据变换与输入输出	- 14 -
5.1 提交 CS_ATOI.C	- 14 -
5.2 提交 CS_ATOF.C	- 14 -
5.3 提交 CS_ITOA.C	- 14 -
5.4 提交 CS_FTOA.C	- 14 -
5.5 讨论分析 OS 的函数对输入输出的数据有类型要求吗	- 14 -
第 6 章 整数表示与运算	- 15 -
6.1 提交 FIB_DG.C	- 15 -
6.2 提交 FIB_LOOP.C	- 15 -
6.3 FIB 溢出验证	- 15 -
6.4 除以 0 验证:	- 15 -
6.5 万年虫验证	- 16 -
6.6 2038 虫验证	- 17 -
第 7 章 浮点数据的表示与运算	- 20 -

7.1 手动 FLOAT 编码:	- 20 -
7.2 特殊 FLOAT 数据的处理.....	- 21 -
7.3 验证浮点运算的溢出	- 21 -
7.4 类型转换的坑.....	- 21 -
7.5 讨论 1: 有多少个 INT 可以用 FLOAT 精确表示	- 22 -
7.6 讨论 2: 怎么验证 FLOAT 采用的向偶数舍入呢	- 22 -
7.7 讨论 3: FLOAT 能精确表示几个 1 元内的钱呢	- 22 -
7.8 FLOAT 的微观与宏观世界	- 22 -
7.9 讨论: 浮点数的比较方法.....	- 23 -
第 8 章 舍尾平衡的讨论	- 24 -
8.1 描述可能出现的问题.....	- 24 -
8.2 给出完美的解决方案.....	- 24 -
第 9 章 总结	- 26 -
9.1 请总结本次实验的收获.....	- 26 -
9.2 请给出对本次实验内容的建议.....	- 26 -
参考文献	- 27 -

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

Char /short int/int/long/float/double/long long/long double/指针

■ 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（linux-x64）

■ 先用递归程序实现，会出现什么问题？

■ 再用循环方式实现。

答：1.递归方式： $f(n) = f(n-1) + f(n-2)$, $n \geq 3$;

$f(1) = f(2) = 1$;

其时间复杂度： $O(2^n)$

2.循环方式：

```
for(int i = 3; i < n; i++) {  
    temp = res;  
    res = pre + res; pre = temp;  
}
```

其时间复杂度： $O(n)$

int 为 47

long int 为 47

unsigned int 为 48

unsigned long 为 48

■ 写出 float/double 类型最小的正数、最大的正数（非无穷）

答：float 最小值： $1.4E-45$

最大值： $3.4028235E38$

double 最小值： $4.9E-324$

最大值： $1.7976931348623157E308$

■ 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

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 安装

CodeBlocks 运行界面截图：编译、运行 hellolinux.c

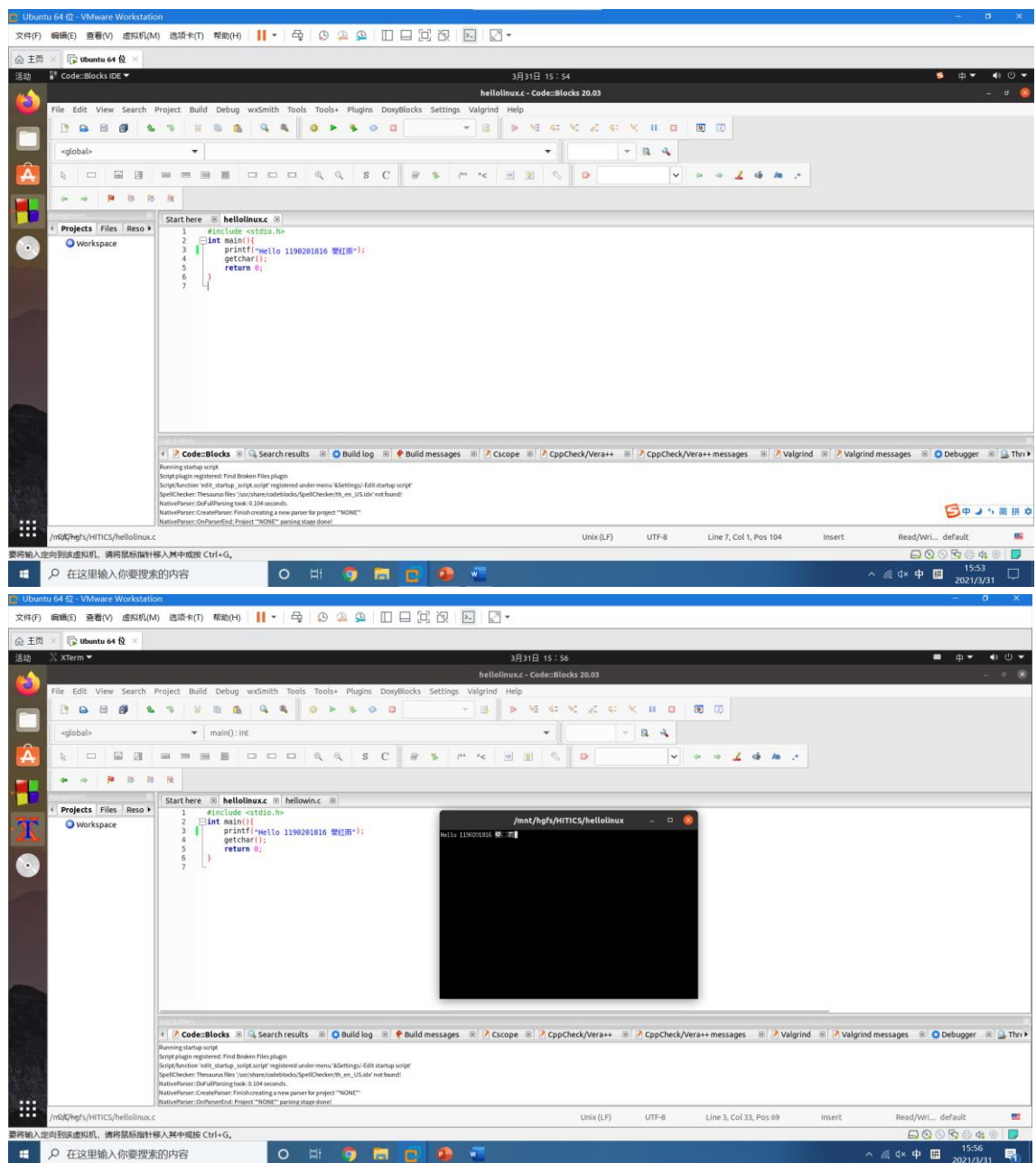


图 2-1 Ubuntu 下 CodeBlocks 截图

2.2 64 位 Ubuntu 下 32 位运行环境建立

在终端下，用 gcc 的 32 位模式编译生成 hellolinux.c。执行此文件。

Linux 及终端的截图。

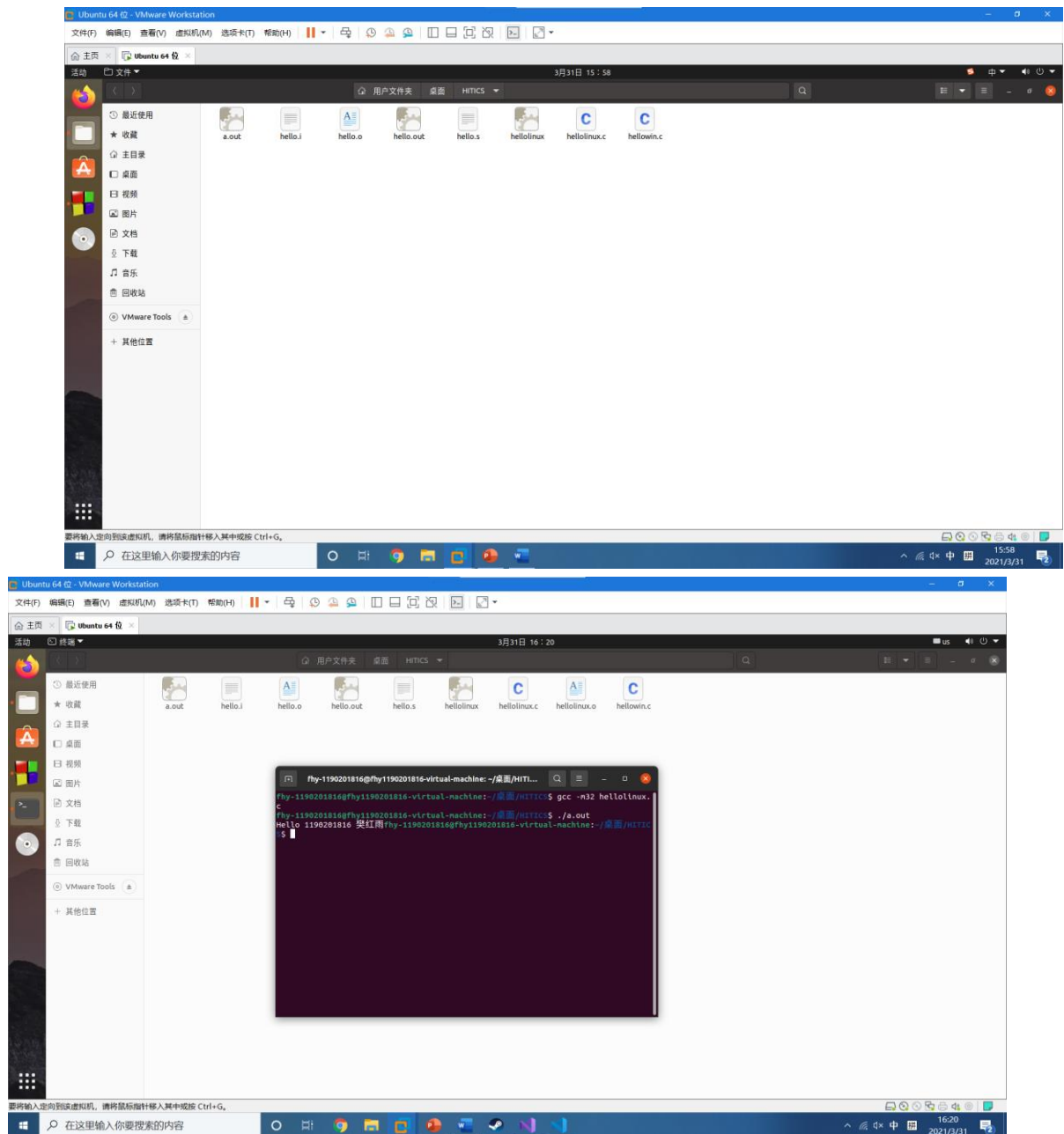


图 2-2 Ubuntu 与 Windows 共享目录截图

第3章 C语言的数据类型与存储

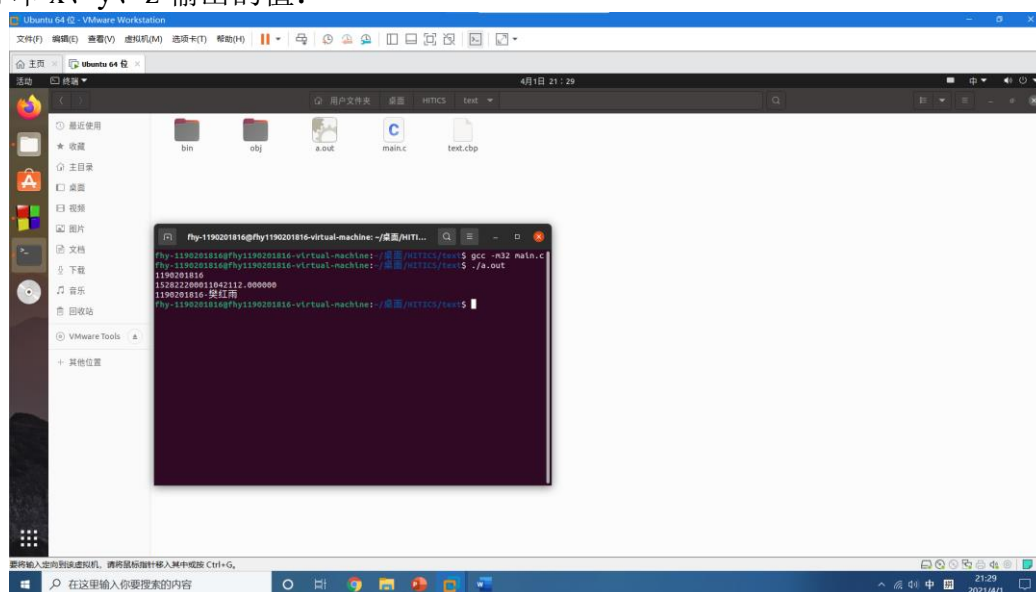
3.1 类型本质

	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	4	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：由编译器来计算，编译阶段就计算出结果了，在运行时就是个常量。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：

```
(gdb) p &x
$3 = (int *) 0x56559020 <x>
(gdb) x /4xb 0x56559020
0x56559020 <x>: 0x28    0xf6    0x0e    0xb9
```

x 的地址：0x56559008 每字节的内容：0x28 0xf6 0x0e 0xb9

```
(gdb) p &y
$4 = (double *) 0xffffffe0
```

```
&y
(double *) 0xffffd5b8
Memory
Address: &y
(e.g. 0x401060, or &variable, or !
0xffffd5b8: 6a 9d 8b d9 77 f7 80 43
```

y 的地址：0xffffd508

每字节的内容：6a 9d 8b d9 77 f7 80 43

```
(gdb) p &z
$5 = (char (*)[100]) 0x56559040 <z>
(gdb) x /20xb &z
0x56559040 <z.0>: 0x31 0x31 0x39 0x30 0x32 0x30 0x31 0x38
0x56559048 <z.0+8>: 0x31 0x36 0x2d 0xe6 0xa8 0x8a 0xe7 0xba
0x56559050 <z.0+16>: 0xa2 0xe9 0x9b 0xa8
```

z 的地址：0x56557010

z 中每个字节的内容：0x31 0x31 0x39 0x 30 0x 32 0x31 0x38 0x31 0x36 0x2d
0xe6 0xa8 0x8a 0xe7 0xba 0xa2 0xe9 0x9b 0xa8

截图 2，标注说明

反汇编查看 x、y、z 在代码段的表示形式。

```
0x565561ed endbr32
0x565561f1 lea ecx,[esp+0x4]
0x565561f5 and esp,0xfffffffff0
0x565561f8 push DWORD PTR [ecx-0x4] → X的代码段
0x565561fb push ebp
0x565561fc mov ebp,esp
0x565561fe push ebx
0x565561ff push ecx
0x56556200 sub esp,0x10
0x56556203 call 0x565560f0 <__x86.get_pc_thunk.bx>
0x56556208 add ebx,0x2dcc
```

```
0x5655620e fld QWORD PTR [ebx-0x1fac] → y的代码段
0x56556214 fstp QWORD PTR [ebp-0x10]
```

```

0x56556217    mov    eax, DWORD PTR [ebx+0x34]
0x5655621d    sub    esp, 0x8
0x56556220    push   eax
0x56556221    lea    eax, [ebx-0x1fcc]
0x56556227    push   eax
0x56556228    call   0x56556080 <printf@plt>
0x5655622d    add    esp, 0x10
;13 :      printf("%f\n", y);
0x56556230    sub    esp, 0x4
0x56556233    push   DWORD PTR [ebp-0xc]
0x56556236    push   DWORD PTR [ebp-0x10]
0x56556239    lea    eax, [ebx-0x1fc8]
0x5655623f    push   eax
0x56556240    call   0x56556080 <printf@plt>
0x56556245    add    esp, 0x10

```

z的代码段

截图 3，标注说明

x 与 y 在__编译__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：数值型常量储存在常量区，而变量储存在动态区，程序运行期间其大小处于动态变化中。处于该区的变量也会时而被创建时而被销毁。

字符串常量与变量在存储空间上的区别是：字符串常量储存在常量区，只可读，而变量储存在静态全局初始化区，和全局变量都储存在数据段，只在定义它的文件中可用，而文件之外是不可以被看见的

常量表达式在计算机中处理方法是：在程序编译时就将值储存在内存中，不可改变，无法直接修改其内容

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，

```

(gdb) p &x
$1 = (int *) 0x56559008 <x>
x 的地址:0x56559008

```

```

&y
(double *) 0xffffd5b8
y 的地址:0xffffd5b8

```

```

(gdb) p &z
$1 = (char (*)[100]) 0x56559040 <z>
z 的地址:0x56559040

```

&argc	(int *) 0xffffd5f0
&argv	(char ***) 0xffffd5f4

argc 的地址: 0xffffd5f0
argv 的地址: 0xffffd5f4

Watches	
Function arguments	
argc	4
argv	0xffffd694

截图 4

argc 的内容: 4

argv 的内容: 0xffffd694

cstr 的地址与内容截图，pstr 的内容与截图，
cstr 的地址：0x565ca040 内容：1190201816-樊红雨
pstr 的地址：0x565ca124 内容：1190201816-樊红雨

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text$ ./str
ctr的内容: 1190201816-樊红雨   ctr的地址: 0x565ca040
ptr的内容: 1190201816-樊红雨   ptr的地址: 0x565ca124
```

```
#include <stdio.h>
#include <string.h>

int x = -1190201816;

char cstr[100] = "1190201816-粉红雨";
char *pstr = "1190201816-粉红雨";
int main( )
{
    double y = 152822200011042111;
    static char z[100] = "1190201816-粉红雨";
    strcpy(cstr, "152822200011042111");
    strcpy(pstr, "152822200011042111");
    printf("cstr的内容: %s  cstr的地址: %p\n", cstr, &cstr);
    printf("pstr的内容: %s  pstr的地址: %p\n", pstr, &pstr);
}
```

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text$ gcc -m32 Ma
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text$ ./a.out
段错误 (核心已转储)
```

(将 pstr 与 cstr 全部修改出现的问题)

```

#include <stdio.h>
#include <string.h>

int x = -1190201816;

char cstr[100] = "1190201816-樊红雨";
char *pstr = "1190201816-樊红雨";
int main( )
{
    double y = 152822200011042111;
    static char z[100] = "1190201816-樊红雨";
    strcpy(cstr, "152822200011042111");
    //strcpy(pstr, "152822200011042111");
    printf("cstr的内容: %s  cstr的地址: %p\n", cstr, &cstr);
    printf("pstr的内容: %s  pstr的地址: %p\n", pstr, &pstr);
    /*
fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITICS/text$ ./a.out
cstr的内容: 152822200011042111  cstr的地址: 0x56629040
pstr的内容: 1190201816-樊红雨  pstr的地址: 0x56629124

```

(只修改 cstr 未出现问题)

```

#include <stdio.h>
#include <string.h>

int x = -1190201816;

char cstr[100] = "1190201816-樊红雨";
char *pstr = "1190201816-樊红雨";
int main( )
{
    double y = 152822200011042111;
    static char z[100] = "1190201816-樊红雨";
    //strcpy(cstr, "152822200011042111");
    strcpy(pstr, "152822200011042111");
    printf("cstr的内容: %s  cstr的地址: %p\n", cstr, &cstr);
    printf("pstr的内容: %s  pstr的地址: %p\n", pstr, &pstr);
    /*
fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITICS/text$ gcc -M32 -m32 -c *.c -o a.out
fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITICS/text$ ./a.out
段错误 (核心已转储)

```

(只修改 pstr 出现错误)

截图 5

pstr 修改内容会出现什么问题: 用学号-姓名初始化 pstr 后, 使用 strcpy 函数修改 pstr 将会产生段错误 (核心已转储)。因为原先 pstr 的值 (学号+姓名) 存储在常量区, 这种赋值语句会意图修改常量区的值, 不合法。

第 4 章 深入分析 UTF-8 编码

4.1 提交 utf8len.c 子程序

4.2 C 语言的 strcmp 函数分析

分析论述：strcmp 到底按照什么顺序对汉字排序

每个汉字都有其对应的 unicode 编码，strcmp 按照汉字对应的 unicode 编码大小对汉字进行排序。所以用 strcmp 比较姓名的大小时，首先比较姓的 unicode 编码大小，若一样，则继续比较下一位名的 unicode 编码大小。

4.3 讨论：按照姓氏笔画排序的方法实现

分析论述：应该怎么实现呢？

建立一个汉字与其笔画顺序对应的索引，即通过汉字可以得到它的笔画顺序。每次排序时，利用该索引中汉字的笔画数量进行排序，若笔画数量相同的姓氏，则按照汉字对应的 unicode 码进行排序，最终实现按照姓氏笔画排序的方法。

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。

两个函数为：`ssize_t read(int fd,void *buf,size_t n)`

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

`read` 函数有一个 `size_t` 的输入参数和一个 `ssize_t` 的返回值。

在 x86-64 系统中，`ssize_t` 被定义为 `unsigned long`,而 `size_t`(有符号的大小)被定义为 `long`。

`read` 函数返回一个有符号的大小，而不是一个无符号大小，这是因为出错时它必须 返回-1

第 6 章 整数表示与运算

6.1 提交 fib_dg.c

6.2 提交 fib_loop.c

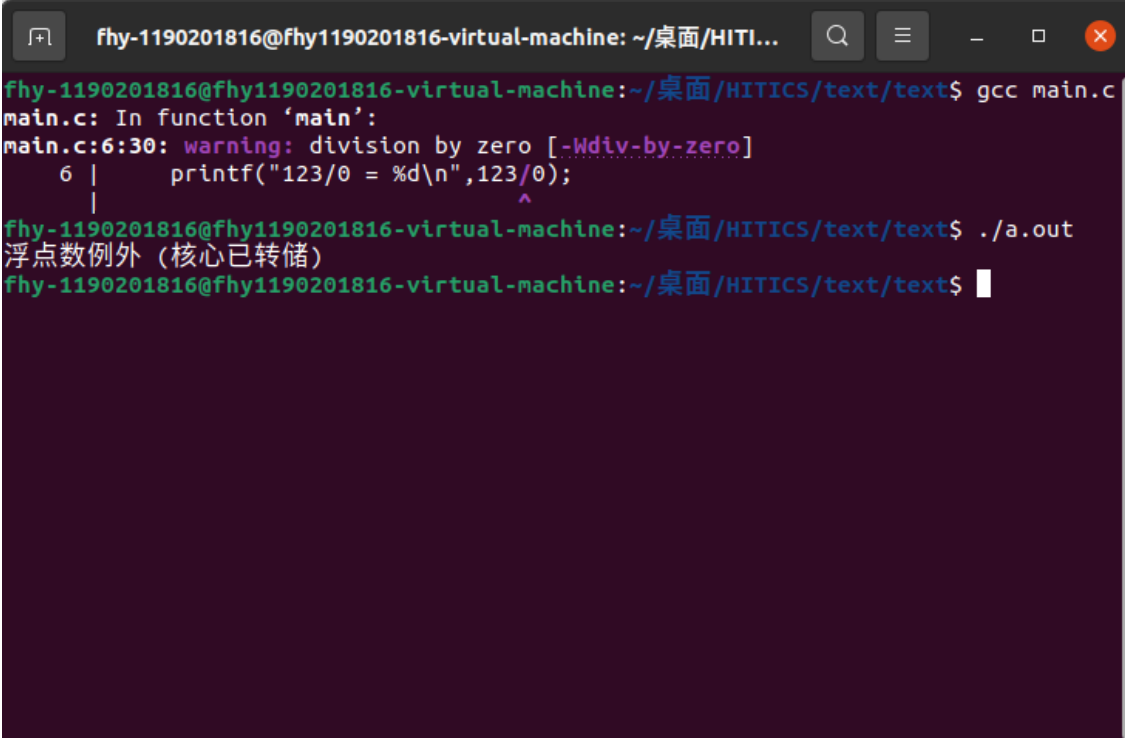
6.3 fib 溢出验证

int 时从 n=__47__时溢出, long 时 n=__93__时溢出。

unsigned int 时从 n=__48__时溢出, unsigned long 时 n=__94__时溢出。

6.4 除以 0 验证:

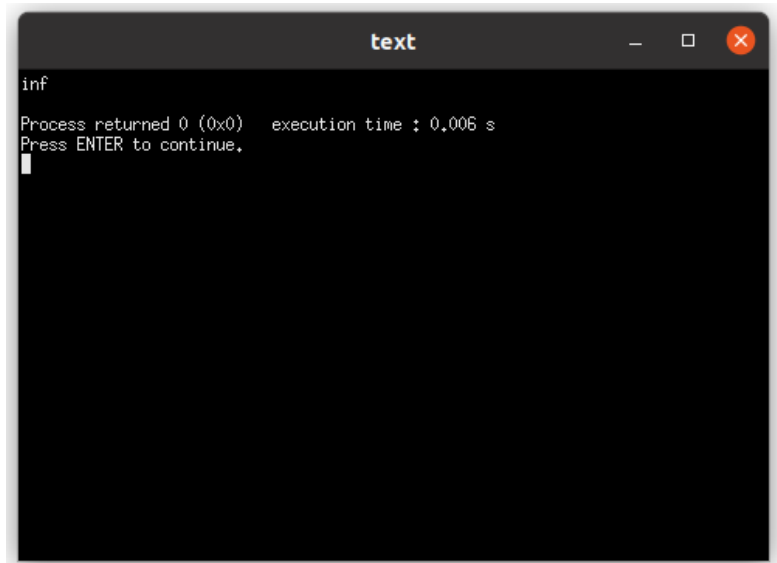
除以 0:

A terminal window with a dark background and light-colored text. The window title is 'fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITI...'. The terminal shows the following commands and output:

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text/text$ gcc main.c
main.c: In function 'main':
main.c:6:30: warning: division by zero [-Wdiv-by-zero]
     6 |     printf("123/0 = %d\n",123/0);
       |                               ^
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text/text$ ./a.out
浮点数例外 (核心已转储)
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text/text$
```

截图 1

除以极小浮点数, 截图:

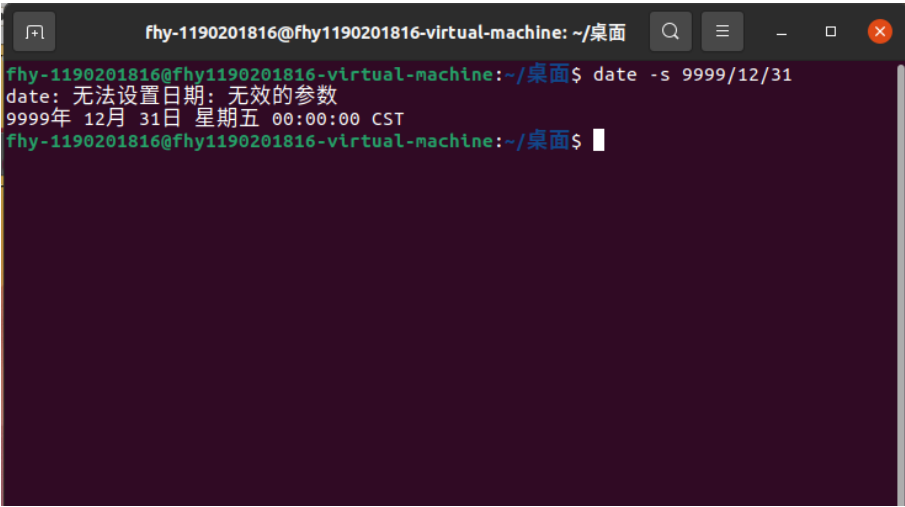


6.5 万年虫验证

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:



我的 Windows 系统无法使用 Bios 修改时间到 9999 年,最多只能到 2099 年。

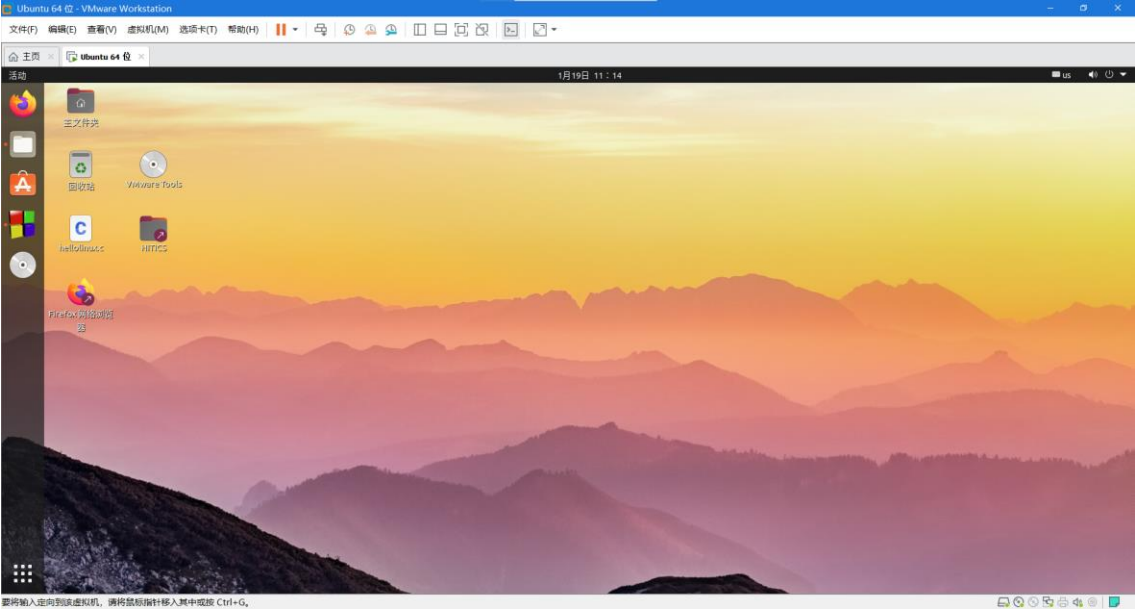
A terminal window titled 'fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面'. The prompt is 'fhy-1190201816@fhy1190201816-virtual-machine:~/桌面\$'. The user has entered the command 'date -s 9999/12/31'. The output is 'date: 无法设置日期: 无效的参数' followed by '9999年 12月 31日 星期五 00:00:00 CST'. The prompt is now 'fhy-1190201816@fhy1190201816-virtual-machine:~/桌面\$' with a cursor.

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面$ date -s 9999/12/31
date: 无法设置日期: 无效的参数
9999年 12月 31日 星期五 00:00:00 CST
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面$
```

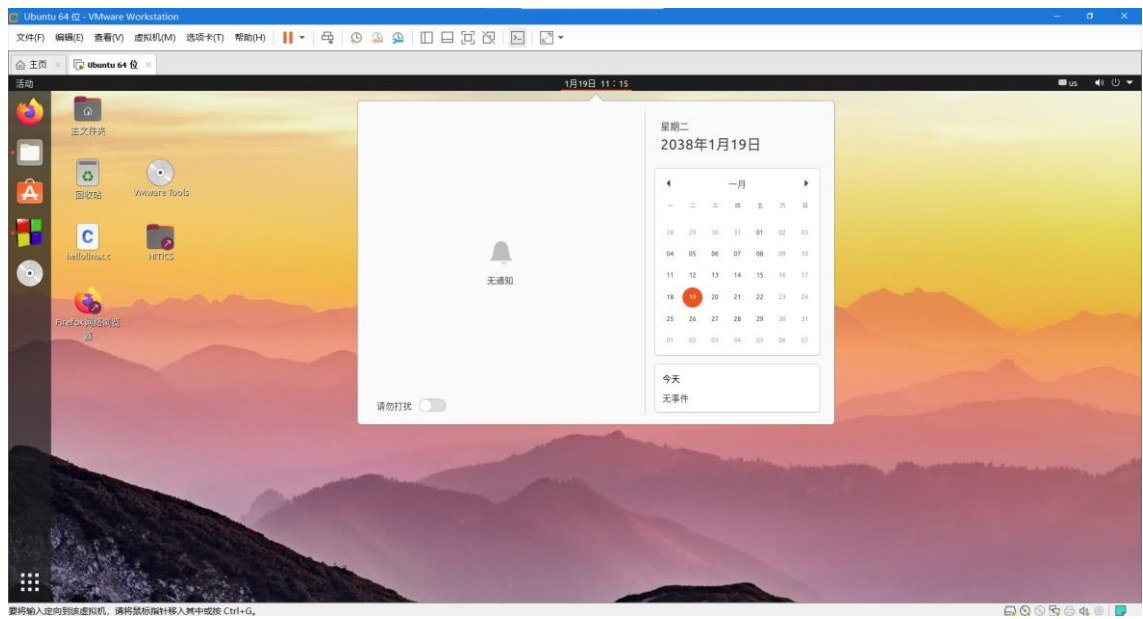
Linux 下使用 date 操作无法修改为 9999.

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）。
编写程序在内存验证手动编码的正确性，截图。

-10.1

负数 $S=1$

$$\begin{aligned} -10.1 &= -1010.00011001100110011001101_{(2)} \\ &= -1.01000011001100110011001101_{(2)} \times 2^3 \end{aligned}$$

向偶数舍入

尾数: $M = -1.01000011001100110011010_{(2)}$

$frac = 01000011001100110011000$

阶码: $E = 3$

$Bias = 127$

$E = exp - Bias$

$exp = 130 = 10000010_{(2)}$

编码结果:

11000001001000011001100110011010

十六进制:

C1 21 99 9a

从低地址到高地址:

9a 99 21 C1

Memory

Address:

(e.g. 0x401060, or &variable)

0x7fffffff42c: 9a 99 21 c1

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。

[illegible]

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

```
fhy-1190201816@fhy1190201816-virtual-machine: ~/桌面/HITI...  
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text/text$ gcc main.c  
main.c: In function 'main':  
main.c:7:30: warning: division by zero [-Wdiv-by-zero]  
    7 |         printf("除以0: %f\n", f/0);  
      |         ^  
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text/text$ ./a.out  
除以0: inf  
除以极小: inf  
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/text/text$
```

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= -1190201816 ，现 x= -1190201856

7.5 讨论 1：有多少个 int 可以用 float 精确表示

有__ 150994944 __个 int 数据可以用 float 精确表示。

是哪些数据呢？__绝对值小于等于 2^{24} 次方或者大于 2^{24} 次方且除最高的 24 位末尾都是 0 的数。

7.6 讨论 2：怎么验证 float 采用的向偶数舍入呢

基于上个讨论，开发程序或举几个特例用 C 验证即可！

截图与标注说明！

```
float x = 18458500; //1000110011010011110000100
float y = 18458499; //1000110011010011110000011
float z = 18458497; //1000110011010011110000001
printf("18458500 = %f\n", x);
printf("18458499 = %f\n", y);
printf("18458497 = %f\n", z);
return 0;
```

Microsoft Visual Studio 调试控制台

```
18458500 = 18458500.000000
18458499 = 18458500.000000
18458497 = 18458496.000000
```

观察 x, y, z 的末尾，即第 25 位，在浮点数的表示中，该位即为需要舍入的位。当该位为 0 时，直接舍入，当该位为 1 时，若前一位为 1 则，进一位，若前一位为 0，则直接舍入。

7.7 讨论 3：float 能精确表示几个 1 元内的钱呢

人民币 0.01-0.99 元之间的十进制数，有多少个可用 float 精确表示？

是哪些呢？

0.25 0.50 0.75

7.8 Float 的微观与宏观世界

按照阶码区域写出 float 的最大密度区域的范围及其密度，最小密度区域及其密度（区域长度/表示的浮点个数）： $-(2-2^{-23}) \times 2^{-126} \sim (2-2^{-23}) \times 2^{-126}$ 、 2^{-149} 、 $-(2-2^{-23}) \times 2^{127} \sim 2^{127}$ 和 $2^{127} \sim (2-2^{-23}) \times 2^{127}$ 、 2^{104}

微观世界：能够区别最小的变化__ 2^{-149} __，其 10 进制科学记数法为 1.401298×10^{-45} __

宏观世界：不能区别最大的变化__ 2^{104} __，其 10 进制科学记数法为__ 2.028241×10^{31} __

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。

任意两个浮点是不能用`==`直接比较。而是通过相减来比较。

如浮点数 a , b 。想要比较 a , b 的大小。则 `fabs(a-b) <= 1e-8` 即为在精度为 $1e-8$ 的情况下表示 `a==b`。

这样是因为有些数无法用浮点数精确表示，发生舍入时可能无法正确判断浮点数的大小关系。

转化为 IEEE 型的浮点数比较方法：

1. 首先判断是否为 NaN。
2. 再判断两个数的符号位，如果一个数为正数，一个负数，则正数大于负数。
例外：正 0 与负 0 相等。
3. 然后比较阶码。若为正数，则阶码大的数大；若为负数阶码小的数大。
4. 最后比较尾数。若为正数，则尾数大的数大；若为负数尾数小的数大。

第 8 章 舍尾平衡的讨论

8.1 描述可能出现的问题

由于报表在统计、汇总时，比如从“元”变成“万元”等等，报表数据就要进行设为取整。通常需要对数据进行四舍五入的操作，这时就会产生误差，而如果报表中右这些数据的合计数值，呢么舍位时产生的误差就会累计，有可能导致舍位过的数据预期合计值无法匹配。例如， $1.5+1.5=3.0$ ，而四舍五入只保留整数部分后，平衡关系就变成了 $2+2=4$ ，这样看上去是错误的。

举例说明：

13,450 元+45,000 元+45,561 元 = 104,011 元。

当单位变成万元，保留两位小数，根据四舍五入的原则：

1.35 万元+4.5 万元+4.6 万元=10.45 万元

出现 0.05 万元的误差。平衡被打破。

8.2 给出完美的解决方案

方案 1：

将平衡差整理到第一个数据中。

例如一组数据：1.48，1.42，0，0.32，6.48，0.98，1.39。

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	Sum
1.48	1.42	0	0.32	6.48	0.98	1.39	12.07

他们的和 Sum 为：12.07。

现在对每个数据进行四舍五入：1，1，0，0，6，1，1。

B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	Sum'
1	1	0	0	6	1	1	12

然而，将四舍五入后的数据相加得到的和 $=10 \neq 12$ 。

此时产生了误差，我们称为“平衡差”。该例子中平衡差 = 2。

我们将平衡差加入到第一个数据中得到一组新的数据：

C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	Sum'
3	1	0	0	6	1	1	12

3，1，0，6，1，1。此时就消除了平衡差。但这种方法也有缺点。

若数据量较大，平衡差的累计也可能变大，进而会使第一个数据产生非常不合理的偏移结果。

方案 2：

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。

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。若平衡差>0，则需要将最小调整值+1，分别加到绝对值最大的数据上。反之，则需要将最小调整值-1 分别加到绝对值最大的数据上。

例如在上面的例子中，平衡差=2。则需要将两个+1，分别加到绝对值最大的数据：6.48，1.48 上。

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]	Sum'
2	1	0	0	7	1	1	12

调整后的数据变为：2，1，0，0，7，1，1。

在这种方案中，平衡差由多个数据分担，而选择绝对值大的数据可以使数据的相对变动最小。在结果中 1.48 舍位后变成了 2，6.48 舍位后变成了 7。从数据上来看这种方法还是比较理想的。

但该方案的缺点也很明显，我们需要找到 k 个绝对值最大的数据，如果数据量较大，则会浪费时间，为了解决这个问题，有方案 3。

方案 3:

将平衡差按照“最小调整值”，依次对不为“0”的数据进行分担。

和上一种平衡的方式类似，该方法将平衡差依次分配给数据。考虑到在四舍五入时 0 不会产生误差，所以将平衡差按照最小调整值依次分配给各个非零的数据。

在上面的例子中：

E[1]	E[2]	E[3]	E[4]	E[5]	E[6]	E[7]	Sum'
2	2	0	0	6	1	1	12

平衡差=2，将两个+1 依次分配给 A[1]=1.48 与 A[2]=1.42，最终变成了 E[1]=2, E[2]=2。调整的结果也比较合理，同时这种方案避免了对于一组数据求 K 个绝对值最大造成的时间浪费，效率较高，所以我认为可以使用这种方法进行舍位平衡。

第 9 章 总结

9.1 请总结本次实验的收获

通过此次实验，我学会了各种变量在内存中的地址不同，学会了使用 Code:Blocks 查看代码的反汇编，知道了反汇编的一些操作。学会了在 Linux 下使用 gcc 编译，并且使用 objdump 和 gdb 进行反汇编的查看。知道了汉字在 UTF-8 的存储方式。

9.2 请给出对本次实验内容的建议

希望老师实验 ppt 的内容可以写的更加详细，有些句子比较难懂。

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

- [1] 林来兴. 空间控制技术[M]. 北京: 中国宇航出版社, 1992: 25-42.
- [2] 辛希孟. 信息技术与信息服务国际研讨会论文集: A 集[C]. 北京: 中国科学出版社, 1999.
- [3] 赵耀东. 新时代的工业工程师[M/OL]. 台北: 天下文化出版社, 1998 [1998-09-26]. <http://www.ie.nthu.edu.tw/info/ie.newie.htm> (Big5) .
- [4] 湛颖. 空间交会控制理论与方法研究[D]. 哈尔滨: 哈尔滨工业大学, 1992: 8-13.
- [5] KANAMORI H. Shaking Without Quaking[J]. Science, 1998, 279 (5359): 2063-2064.
- [6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science , 1998 , 281 : 331-332[1998-09-23]. <http://www.sciencemag.org/cgi/collection/anatmorp>.