

实验报告

实 验（二）

题 目 DataLab 数据表示

专 业 计算学部

学 号 1190200208

班 级 1936002

学 生 李旻翀

指 导 教 师 王忠杰

实 验 地 点 G709

实 验 日 期 2021.4.1

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	- 7 -
2.1 UBUNTU 下 CODEBLOCKS 安装	- 7 -
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	- 9 -
3.1 类型本质（1 分）	- 9 -
3.2 数据的位置-地址（2 分）	- 9 -
3.3 MAIN 的参数分析（2 分）	- 11 -
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.....	- 15 -
6.6 2038 虫验证	- 16 -
第 7 章 浮点数据的表示与运算.....	- 17 -
7.1 手动 FLOAT 编码:	- 17 -
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.....	- 19 -
7.6 讨论 2: 怎么验证 FLOAT 采用的向偶数舍入呢	- 19 -
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.....	- 20 -
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	- 22 -
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

1. 上实验课前, 必须认真预习实验指导书 (PPT 或 PDF)
2. 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。
3. 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

	Win/VS/x86	Win/VS/x64	Win/CB/32	Win/CB/32	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

4. 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）

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。

5. 先用递归程序实现，会出现什么问题？

运行速度太慢。

6. 再用循环方式实现。

具体程序见附件 fib_loop.c。

7. 写出 float/double 类型最小的正数、最大的正数（非无穷）

float 类型最小的正数：0 00000000 0000 0000 0000 0000 0000 001

float 类型最大的正数：0 11111110 1111 1111 1111 1111 1111 111

double 类型最大的正数：1.79769*10⁽³⁰⁸⁾

double 类型最小的正数：2.22507*10⁽⁻³⁰⁸⁾

8. 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

整数部分：10，即二进制数 1010

小数部分：0.1，即二进制数 0.00011001100110011001100110011

用二进制科学计数法表示：

1.010 000 1100 1100 1100 1100 1100...

故指数部分 $e = E + \text{bias} = 3 + 127 = 130 = 10000010 (2)$

小数部分 $f = M - 1 = 0100 0011 0011 0011 0011 010$

(保留 23 位，最后两位涉及舍入)

故用 IEEE 754 标准表示该 float:

1 1000 0010 0100 0011 0011 0011 0011 010

即：

1100 0001 0010 0001 1001 1001 1001 1010

转化为 16 进制，即为：

c1 21 99 9a

所以从低到高字节排序为：

9a 99 21 c1

9. 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（表示的浮点数个数/区域长度）

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）：

____ $-(2-2^{-23}) \cdot 2^{-126} \sim (2-2^{-23}) \cdot 2^{-126}$ ____、 ____ $(2-2^{-23}) \cdot 2^{-150}$ ____、
 ____ $2^{127} \sim (2-2^{-23}) \cdot 2^{127}$ 和 ____ $-(2-2^{-23}) \cdot 2^{127} \sim -2^{127}$ ____、
 ____ $(2-2^{-23}) \cdot 2^{103}$ ____

微观世界：能够区别最小的变化 ____ $(2-2^{-23}) \cdot 2^{-150}$ ____，其 10 进制科学记数法为 ____ $1.401298 \cdot 10^{-45}$ ____

宏观世界：不能区别最大的变化 ____ $(2-2^{-23}) \cdot 2^{103}$ ____，其 10 进制科学记数法为 ____ $2.028241 \cdot 10^{31}$ ____

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

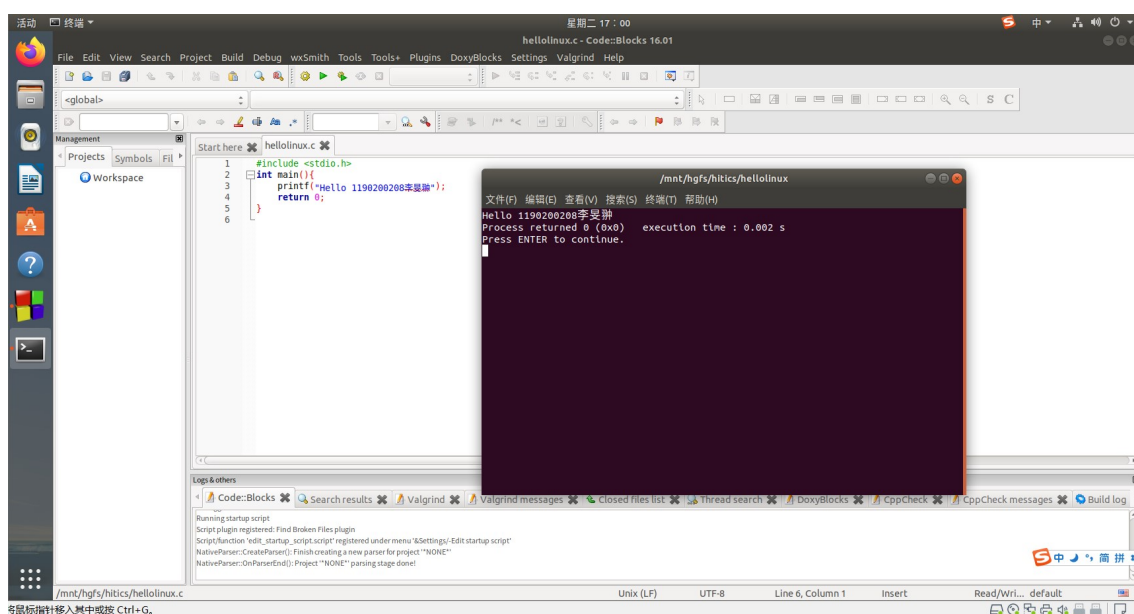


图 2-1 Ubuntu 下 CodeBlocks 截图

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。

计算机系统实验报告

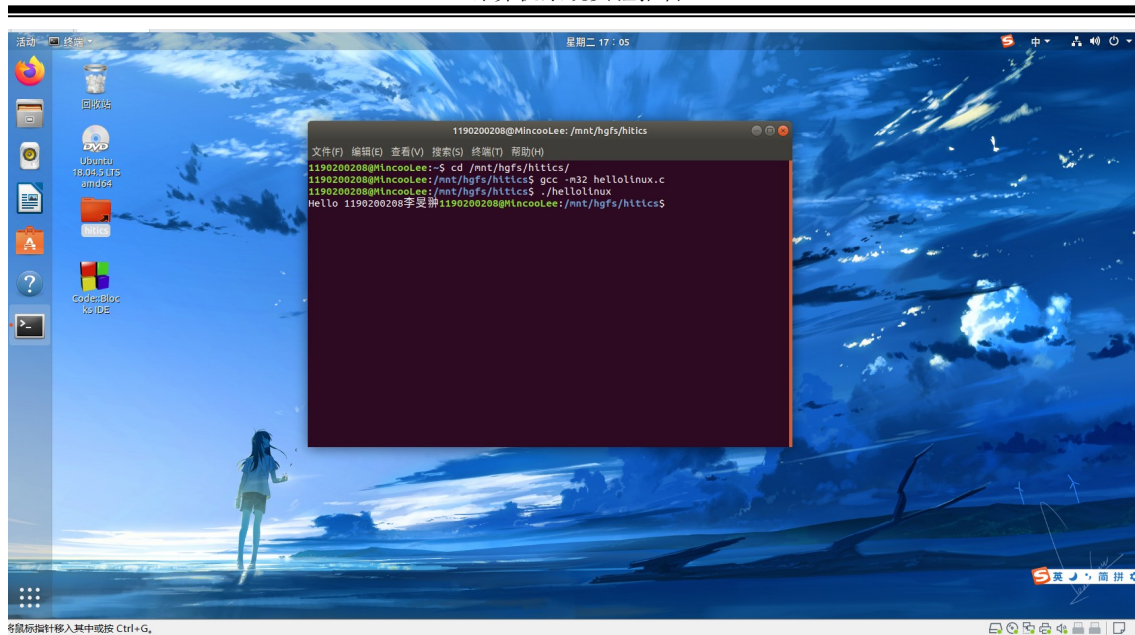


图 2-2 Ubuntu 与 Windows 共享目录截图

第 3 章 C 语言的数据类型与存储

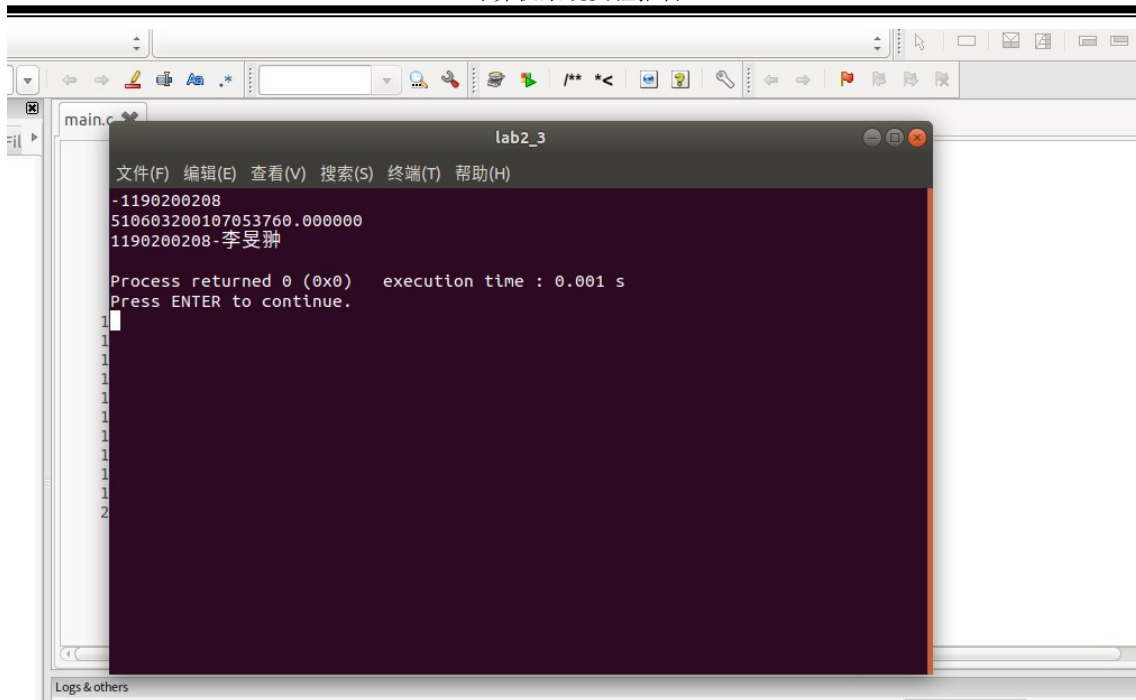
3.1 类型本质

	Win/VS/x86	Win/VS/x64	Win/CB/32	Win/CB/32	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：Sizeof 不是函数，而是关键字。对于每个数据类型都有一个固定的 `align size`，即类型大小。在编译时，编译器根据代码确定 `sizeof` 返回的数据类型，根据数据类型将 `sizeof` 表达式直接替换为对应的常量 `align size`。

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

```
(qdb) x /4xb &x
0x56557020 <x>: 0x70 0xfc 0x0e 0xb9
(qdb) x /4xb &y
0xffffd010 0x0b 0x3c 0xee 0x9e
(qdb) x /20xb &z
0x56557040 <z.2384>: 0x31 0x31 0x39 0x30 0x32 0x30 0x30 0x32
0x56557048 <z.2384+8>: 0x30 0x38 0x2d 0xe6 0x9d 0xe6 0xe6 0x97
0x56557050 <z.2384+16>: 0xbb 0xe7 0xbf 0x80
(qdb)
```

红框内为地址
蓝框内为每字节的内容

x,y,z 的地址与每字节的内容（通过设置断点，在程序运行过程中查看）

反汇编查看 x、y、z 在代码段的表示形式。截图 3，标注说明

```
Disassembly of section .data:

00000000 <x>:
 0: 70 fc
 2: 0e
 3: b9 00 00 00
 ...

00000020 <z.2384>:
20: 31 31
22: 39 30
24: 32 30
26: 30 32
28: 30 38
2a: 2d e6 9d 8e e6
2f: 97
30: bb e7 bf 80 00
 ...
```

```
jo      ffffffff <z.2384+0xfffffde>
push    %cs
mov     $0x0,%ecx

xor     %esi,(%ecx)
cmp     %esi,%eax
xor     (%eax),%dh
xor     %dh,(%edx)
xor     %bh,(%eax)
sub     $0xe68e9de6,%eax
xchg    %eax,%edi
mov     $0x80bfe7,%ebx
```

x对应的代码段
z对应的代码段

```

00000000 <.rodata>:
 0: 25 64 0a 00 25      and    $0x25000a64,%eax
 5: 6c                  insb   (%dx),%es:(%edi)
 6: 66 0a 00            data16 or    (%eax),%al
 9: 00 00              add    %al,(%eax)
 b: 00 78 5f           add    %bh,0x5f(%eax)
 e: 61                popa
 f: 64 64 72 65        fs fs  jb  78 <z.2384+0x58>
13: 73 73              jae   88 <z.2384+0x68>
15: 3a 25 70 0a 79 5f   cmp    0x5f790a70,%ah
1b: 61                popa
1c: 64 64 72 65        fs fs  jb  85 <z.2384+0x65>
20: 73 73              jae   95 <z.2384+0x75>
22: 3a 25 70 0a 7a 5f   cmp    0x5f7a0a70,%ah
28: 61                popa
29: 64 64 72 65        fs fs  jb  92 <z.2384+0x72>
2d: 73 73              jae   a2 <z.2384+0x82>
2f: 3a 25 70 0a 00 00   cmp    0xa70,%ah
35: 00 00              add    %al,(%eax)
37: 00 0b              add    %cl,(%ebx)
39: 3c ee              cmp    $0xee,%al
3b: 9e                sahf
3c: 1b 58 9c           sbb    -0x64(%eax),%ebx

```

通过 objdump -d xxx.o 命令进行反汇编查看。其中，x 与 z 的代码段可以在.data 中直接看到，而 y 的代码段需要到.rodata 中根据其值进行寻找。

图中红色框标注了 y 对应的内容（00 0b, 3c ee, 9e）和 y 对应的代码段（add %cl,(%ebx), cmp \$0xee,%al, sahf）。红色箭头指向了 y 对应的代码段。

通过 `objdump -d xxx.o` 命令进行反汇编查看。其中，x 与 z 的代码段可以在 .data 中直接看到，而 y 的代码段需要到 .rodata 中根据其值进行寻找。

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：__宏常量不进行存储，const 常量存储在代码区，静态变量和全局变量存储在数据区，局部变量存储在堆栈区__

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，不可修改。而对于字符串变量，静态变量和全局变量保存在数据区，局部变量在堆栈区__

常量表达式在计算机中处理方法是：__在编译阶段就转换成常量，存储在内存中，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，截图 4

```

(gdb) p *(argv+1)
$1 = 0xffffd2b6 "x"
(gdb) p *(argv+2)
$2 = 0xffffd2b8 "y"
(gdb) p *(argv+3)
$3 = 0xffffd2ba "z"
(gdb) p &argc
$4 = (int *) 0xffffd020
(gdb) p &argv
$5 = (char ***) 0xffffcfec
(gdb) p argv
$6 = (char **) 0xffffd0b4
(gdb)

```

图中红色括号标注了 x,y,z 的地址（\$1, \$2, \$3）和 argc,argv 的地址及 argv 的值（\$4, \$5, \$6）。红色箭头指向了 x,y,z 的地址。

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

```

8      {
9          printf("%p\n",cstr);
10         printf("%p\n",pstr);
11
12         return 0;
13     }
(gdb) b 11
Breakpoint 1 at 0x579: file main.c, line 11.
(gdb) r
Starting program: /home/1190200208/CB/lab2_4/main
0x56557020
0x56555610

Breakpoint 1, main () at main.c:12
12         return 0;
(gdb) x /20xb 0x56557020
0x56557020: <cstr>: 0x31 0x31 0x39 0x30 0x32 0x30 0x30 0x32
0x56557028: <cstr+8>: 0x30 0x38 0x2d 0xe6 0x9d 0x8e 0xe6 0x97
0x56557030: <cstr+16>: 0xbb 0xe7 0xbf 0x80
(gdb) x /20xb 0x56555610
0x56555610: 0x31 0x39 0x30 0x32 0x30 0x30 0x32 0x30
0x56555618: 0x38 0x2d 0xe6 0x9d 0x8e 0xe6 0x97 0xbb
0x56555620: 0xe7 0xbf 0x80 0x00
(gdb)

```

Diagram annotations in the image:

- Red arrow pointing to `0x56557020`: **cstr的地址**
- Red arrow pointing to `0x31 0x31 0x39 0x30 0x32 0x30 0x30 0x32`: **cstr的内容**
- Red arrow pointing to `0x56555610`: **pstr的地址**
- Red arrow pointing to `0x31 0x39 0x30 0x32 0x30 0x30 0x32 0x30`: **pstr的内容**

pstr 修改内容会出现什么问题____字符串指针为不可修改的常量，修改会导致程序报错_____

第 4 章 深入分析 UTF-8 编码

4.1 提交 utf8len.c 子程序

随压缩包一并提交

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

strcmp()函数的本质是比较二进制数值，所以在 Linux 系统下，strcmp()实际上比较的是不同汉字之间的 unicode 码值，由此进行排序。若比较的是汉字字符串，则是由左到右一位一位比较，直到遇到 unicode 码值不相同的位为止。

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？

将每个姓氏对应的笔画数量进行编码，存储在一个数据库中。每次进行姓氏笔画排序时，都调用这个数据库，按笔画数量对姓氏进行排序。若姓氏的笔画数量相同，则考虑可以比较不同姓氏间的 unicode / UTF-8 编码值来排序。

第 5 章 数据变换与输入输出

5.1 提交 `cs_atoi.c`

随压缩包一并提交

5.2 提交 `cs_atof.c`

随压缩包一并提交

5.3 提交 `cs_itoa.c`

随压缩包一并提交

5.4 提交 `cs_ftoa.c`

随压缩包一并提交

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：

OS 的输入输出 `read/write` 函数对输入输出的数据类型有要求。

C 语言的 `read/write` 函数都位于 `<unistd.h>` 中，具体形式分别为：

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

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

两函数均有一个 `size_t` 类型的输入参数和一个 `ssize_t` 类型的返回值。在 x86-64 系统中，`size_t` 被定义为 `unsigned long`，而 `ssize_t` (有符号的大小) 被定义为 `long`。这是因为函数运行出错时返回 -1。故函数需要返回一个有符号数。

第 6 章 整数表示与运算

6.1 提交 fib_dg.c

随压缩包一并提交

6.2 提交 fib_loop.c

随压缩包一并提交

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 0 时, 程序无法运行



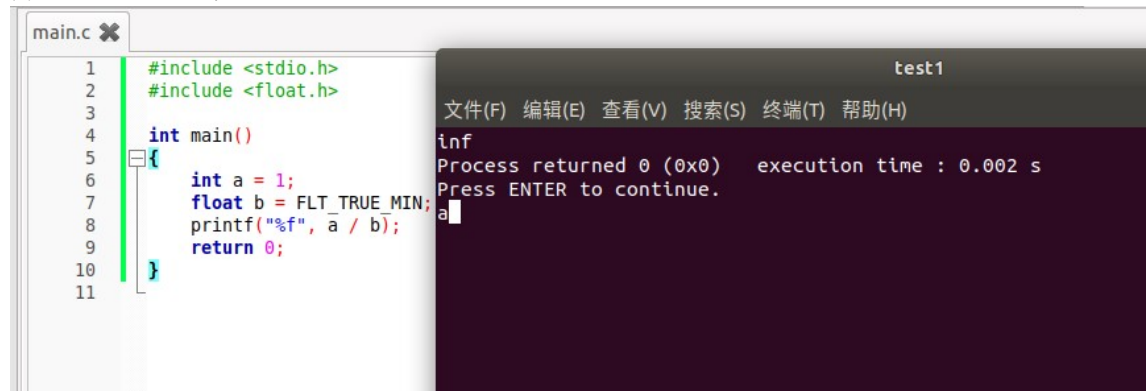
The screenshot shows a code editor with a file named 'main.c' containing the following code:

```
1 #include <stdio.h>
2
3
4 int main()
5 {
6     int a = 1;
7     printf("%d", a / 0);
8     return 0;
9 }
10
```

To the right of the code editor is a terminal window titled 'test1'. It displays the following output:

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Floating point exception (core dumped)
Process returned 136 (0x88)   execution time : 0.155 s
Press ENTER to continue.
```

除以极小浮点数, 截图:



The screenshot shows a code editor with a file named 'main.c' containing the following code:

```
1 #include <stdio.h>
2 #include <float.h>
3
4 int main()
5 {
6     int a = 1;
7     float b = FLT_TRUE_MIN;
8     printf("%f", a / b);
9     return 0;
10 }
11
```

To the right of the code editor is a terminal window titled 'test1'. It displays the following output:

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
inf
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

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:

选做, 未做

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

选做, 未做

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

整数部分：10，即二进制数 1010

小数部分：0.1，即二进制数 0.0001100110011001100110011

用二进制科学计数法表示：

$1.010\ 000\ 1100\ 1100\ 1100\ 1100\ 1100\dots \times 2^3$

故指数部分 $e = E + \text{bias} = 3 + 127 = 130 = 10000010_{(2)}$

小数部分 $f = M - 1 = 0100\ 0011\ 0011\ 0011\ 0011\ 010$

(保留 23 位，最后两位涉及舍入)

故用 IEEE 754 标准表示该 float：

1 1000 0010 0100 0011 0011 0011 0011 010

即：

1100 0001 0010 0001 1001 1001 1001 1010

转化为 16 进制，即为：

c1 21 99 9a

所以从低到高字节排序为：

9a 99 21 c1

编写程序在内存验证手动编码的正确性，截图。

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

typedef unsigned char *byte_pointer;

void show_bytes(byte_pointer start, size_t len)
{
    size_t i;
    for (i = 0; i < len; i++)
    {
        printf(" %.2x", start[i]);
    }
    printf("\n");
}

void show_float(float x)
{
    show_bytes((byte_pointer)&x, sizeof(float));
}

int main()
{
    //数据初始化
    float f = -10.1;
    //测试
    printf(
        "变量名          内容
    );
    printf(
```

变量名	内容	地址	十六进制内存
f	-10.100000	000000000061FE18	9a 99 21 c1
(base) PS_E:\Programming\C\CSAPP\lab2>			

7.2 特殊 float 数据的处理

提交子程序 floatx.c，要求：

子程序随压缩包一并提交

提交子程序 float0.c

子程序随压缩包一并提交

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

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= 314159535 , 现 x= 314159520

有 150994943 个 int 数据可以用 float 精确表示。

是哪些数据呢? ① $-2^{24} \sim 2^{24}$ 间的 int 值 ②绝对值在 $2^{24} \sim 2^{32}$ 且除了最

高 24 位后面的位数均为 0 的

具体计算式为 $(2^{25}-1) + \overline{2^{24}+2^{24}+2^{24}+2^{24}+2^{24}+2^{24}+2^{24}+2^{24}}$

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！

```

int main()
{
    // a = 0b 1000 0000 0000 0000 0000 000 010
    // b = 0b 1000 0000 0000 0000 0000 000 110
    float a = 33554434;
    float b = 33554438;
    show_float(a);
    show_float(b);
    return 0;
}

```

```

00 00 00 4c
02 00 00 4c

```

第一个数字第 24 位是 0，之后多余数字是 10，由于 float 类型尾数只有 23 位，所以需要进行舍入，在 23 位有效数字之后是 10，为半值，而前面的有效数为偶数，所以直接舍去多余位。所以结果为 00 00 00 4c（小端法表示）

第二个数字第 24 位是 1，之后多余数字是 10，根据向偶数舍入的原则，要使最低位为 1 则进位 1，使得最低位仍为 0（偶数），因此源数字变为 02 00 00 4c（小端法表示）。

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 (只考虑小数点后两位的情况)

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）： $-(2-2^{-23}) \times 2^{-126} \sim (2-2^{-23}) \times 2^{-126}$ 、 $-(2-2^{-23}) \times 2^{-150}$ 、 $2^{127} \sim (2-2^{-23}) \times 2^{127}$ 和 $-(2-2^{-23}) \times 2^{127} \sim -2^{127}$ 、 $(2-2^{-23}) \times 2^{103}$

微观世界：能够区别最小的变化 $(2-2^{-23}) \times 2^{-150}$ ，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 $(2-2^{-23}) \times 2^{103}$ ，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。

比较方法：设置一个很小的宏常量作为精度（一般定义为 $1e-7$ ），对于两个浮点数 a, b ，如果 $\text{fabs}(a-b)$ 比精度值要小，则可以认为两个浮点数 a, b 相等。若 a 比 b 大且 $\text{fabs}(a-b)$ 比精度值要大，则认为 a 大于 b 。

理由：因为在计算机中浮点数的存储方式，浮点数在计算机中并非精确存储，而是近似存储，与真实值可能存在误差。因此，在对两个浮点数进行大小比较时，不能简单地使用大于小于号进行比较，而应判断两个浮点数之间的差值是否小于某个值，这里的“某个值”便定义为精度。

第 8 章 舍尾平衡的讨论

8.1 描述可能出现的问题

舍位，就是将数据抹去尾部的一位或几位数，从而达到节省存储空间，或统一格式的作用。但简单直接的舍位处理，如四舍五入，可能会存在潜在的隐患，造成最终统计数据的偏差。

例子：若采用四舍五入进行舍位，则可能出现舍位之后的数据之和与数据求和后再进行舍位的值无法匹配的问题，例如 $2.5 + 2.5 = 5.0$ ，舍位保留整数后值为 5；但若先舍位再求和，则会得到值为 6 的错误结果。

8.2 给出完美的解决方案

1. 提高数据的精度，保留更多的数据位数，减少舍位时舍掉的位数，从而达到减小舍位误差的目的。
2. 保留原始数据位数，并尽量避免出现舍位的情况。例如：在利用表格统计数据时，在需要数据之和、均值等统计数据时才在计算后进行舍位操作。原始数据则不进行舍位操作，这样可以减小舍位误差的累积，使数据更加准确。

第 9 章 总结

9.1 请总结本次实验的收获

通过本次实验，我的收获如下：

1. 对 Linux 下 CodeBlocks 编程、调试的流程更加熟悉
2. 熟悉了 GDB 调试工具的使用，对反汇编等操作有了初步概念
3. 对 Linux 环境下各类变量的生存周期、存储位置有了初步了解
4. 对机器级指令有了一定的了解
5. 对浮点数在计算机中的存储方式有了更深的理解

9.2 请给出对本次实验内容的建议

希望能在一定程度上优化 PPT 内容，现有内容读起来给人造成了理解上的困难；与此同时，希望能够合理安排实验目标。现有的实验任务难度梯度较高，相当部分实验任务还没有学习，在这种情况下完成这些任务让得到正向反馈变成一件不太容易的事[。

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>.