

哈尔滨工业大学

实验报告

实 验（二）

题 目 DataLab 数据表示

专 业 计算机系

学 号 1190201421

班 级 1936603

学 生 张瑞

指 导 教 师 刘宏伟

实 验 地 点 G709

实 验 日 期 2021 年 4 月 1 日

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.....	- 7 -
3.2 数据的位置-地址	- 8 -
3.3 MAIN 的参数分析	- 10 -
3.4 指针与字符串的区别.....	- 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	- 15 -
第 5 章 数据变换与输入输出	- 16 -
5.1 提交 CS_ATOI.C	- 16 -
5.2 提交 CS_ATOF.C	- 16 -
5.3 提交 CS_ITOA.C	- 16 -
5.4 提交 CS_FTOA.C	- 16 -
5.5 讨论分析 OS 的函数对输入输出的数据有类型要求吗	- 16 -
第 6 章 整数表示与运算	- 17 -
6.1 提交 FIB_DG.C.....	- 17 -
6.2 提交 FIB_LOOP.C	- 17 -
6.3 FIB 溢出验证.....	- 17 -
6.4 除以 0 验证.....	- 17 -
6.5 万年虫验证.....	- 18 -
6.6 2038 虫验证	- 18 -
第 7 章 浮点数据的表示与运算	- 19 -

7.1 手动 FLOAT 编码.....	- 19 -
7.2 特殊 FLOAT 数据的处理.....	- 20 -
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	- 23 -
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。

写出 float/double 类型最小的正数、最大的正数（非无穷）。

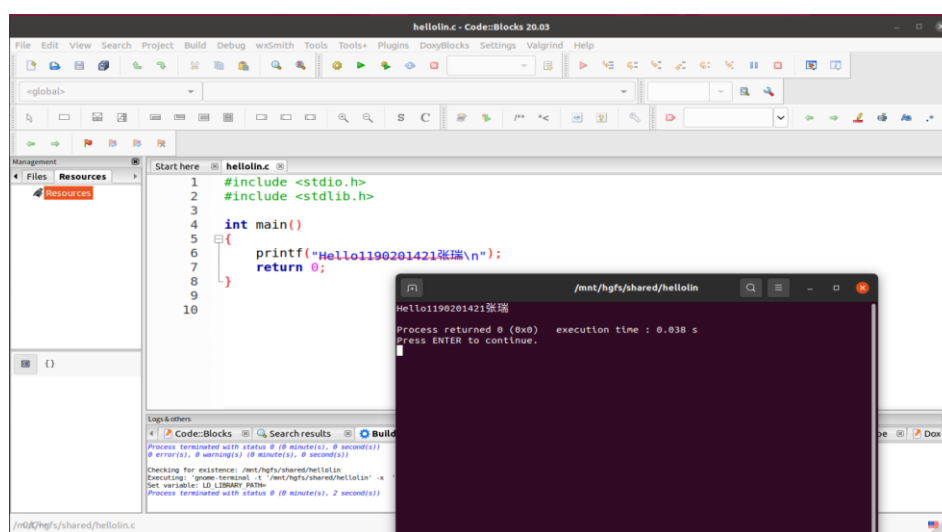
按步骤写出 float 数-10.1 在内存从低到高地址的字节值-16 进制。

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（表示的浮点数个数/区域长度）。

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 安装

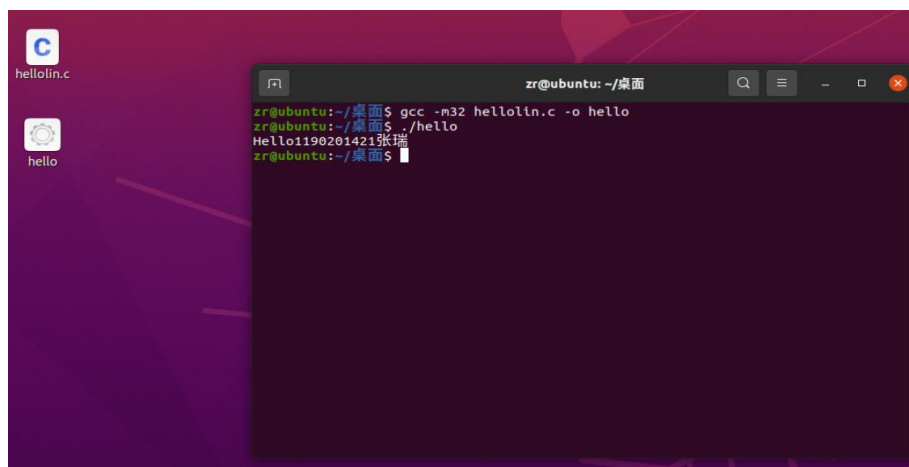
CodeBlocks 运行界面截图：编译、运行 hellolinux.c:



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:



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

Win/VS/x86:

```
char    short    int    long    long long    float    double    long double    pointer
1       2       4       4       8              4       8       8              4
```

Win/VS/x64:

```
char    short    int    long    long long    float    double    long double    pointer
1       2       4       4       8              4       8       8              8
```

Win/CB/64:

```
char    short    int    long    long long    float    double    long double    pointer
1       2       4       4       8              4       8       16             8
```

Win/CB/32:

```
char    short    int    long    long long    float    double    long double    pointer
1       2       4       4       8              4       8       12             4
```

Linux/CB/64:

```
char    short    int    long    long long    float    double    long double    pointer
1       2       4       8       8              4       8       16             8
```

Linux/CB/32:

```
zr@ubuntu:~/code/codeblocks/viewsize$ gcc -m32 main.c -o viewsize
zr@ubuntu:~/code/codeblocks/viewsize$ ./viewsize
char    short    int    long    long long    float    double    long double    pointer
1       2       4       4       8              4       8       12             4
```

C 编译器对 sizeof 的实现方式：从汇编文件和反汇编能看出，在使用 sizeof 时并没有进行函数调用，而是将一个立即数存入寄存器，所以 sizeof 实际并不是一个函数。

main.s

保存(S)

打开(O) 主 代码块/视图
~code/codeblocks/viewsite

```
1 .file "main.c"
2 .text
3 .section .rodata
4 .align 8
5 .LC0:
6 .string "char\t short\t int\t long\t long long\t float\t double\t long double\t pointer"
7 .align 8
8 .LC1:
9 .string "%d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\n"
10 .text
11 .globl main
12 .type main, @function
13 main:
14 .LFB6:
15 .cfi_startproc
16 endbr64
17 pushq %rbp
18 .cfi_def_cfa_offset 16
19 .cfi_offset 6, -16
20 movq %rsp, %rbp
21 .cfi_def_cfa_register 6
22 leaq .LC0(%rip), %rdi
23 call puts@PLT
24 pushq $8
25 pushq $16
26 pushq $8
27 pushq $4
28 movl $8, %r9d
29 movl $8, %r8d
30 movl $4, %ecx
31 movl $2, %edx
32 movl $1, %esi
33 leaq .LC1(%rip), %rdi
34 movl $0, %eax
35 call printf@PLT
36 addq $32, %rsp
37 movl $0, %eax
38 leave
39 .cfi_def_cfa 7, 8
40 ret
41 .cfi_endproc
42 .LFE6:
43 .size main, .-main
44 .ident "GCC: (Ubuntu 9.3.0-17ubuntu1-20.04) 9.3.0"
45 .section .note.GNU-stack,"",@progbits
46 .section .note.gnu.property,"a"
47 .align 8
48 .long 1f - 0f
49 .long 4f - 1f
50 .long 5
51 0:
52 .string "GNU"
53 1:
54 .align 8
55 .long 0xc0000002
56 .long 2f - 2f
```

纯文本 制表符宽度: 8 第 1 行, 第 1 列 插入

```
0000000000001169 <main>:
1169: f3 0f 1e fa      endbr64
116d: 55              push    %rbp
116e: 48 89 e5         mov     %rsp,%rbp
1171: 48 8d 3d 90 0e 00 00 lea     0xe90(%rip),%rdi        # 2008 <_IO_stdin_used+0x8>
1178: e8 e3 fe ff ff   callq   1060 <puts@plt>
117d: 6a 08           pushq   $0x8
117f: 6a 10           pushq   $0x10
1181: 6a 08           pushq   $0x8
1183: 6a 04           pushq   $0x4
1185: 41 b9 08 00 00 00 mov     $0x8,%r9d
118b: 41 b8 08 00 00 00 mov     $0x8,%r8d
1191: b9 04 00 00 00 mov     $0x4,%ecx
1196: ba 02 00 00 00 mov     $0x2,%edx
119b: be 01 00 00 00 mov     $0x1,%esi
11a0: 48 8d 3d a9 0e 00 00 lea     0xea9(%rip),%rdi        # 2050 <_IO_stdin_used+0x50>
11a7: b8 00 00 00 00 mov     $0x0,%eax
11ac: e8 bf fe ff ff   callq   1070 <printf@plt>
11b1: 48 83 c4 20      add     $0x20,%rsp
11b5: b8 00 00 00 00 mov     $0x0,%eax
11ba: c9             leaveq
11bb: c3            retq
11bc: 0f 1f 40 00     nopl    0x0(%rax)
```

3.2 数据的位置-地址

打印 x 、 y 、 z 输出的值：（为防止浮点数超出表示范围，仅采用身份证号后 8 位）


```

zr@ubuntu:~/code/codeblocks/datalocation$ gcc -m32 -g main.c -o datalocation
zr@ubuntu:~/code/codeblocks/datalocation$ ./datalocation
-1190201421
10261225.000000
1190201421-张瑞

```

反汇编查看 x、y、z 的地址，每字节的内容：

x 的地址：0x56559008 每字节内容：0xb3 0xf7 0x0e 0xb9

```

(gdb) p &x
$1 = (int *) 0x56559008 <x>
(gdb) x/4xb &x
0x56559008 <x>: 0xb3    0xf7    0x0e    0xb9

```

y 的地址：0xffffcfcc 每字节内容：0xe9 0x92 0x1c 0x4b

```

(gdb) p &y
$2 = (float *) 0xffffcfcc
(gdb) x/4xb &y
0xffffcfcc:    0xe9    0x92    0x1c    0x4b

```

z 的地址：0x5655900c 每字节内容：0x31 0x31 0x39 0x30 0x32 0x30 0x31 0x34 0x32 0x31 0x2d 0xe5 0xbc 0xa0 0xe7 0x91 0x9e 0x00

```

(gdb) p &z
$3 = (char (*)(18)) 0x5655900c <z>
(gdb) x/18xb &z
0x5655900c <z.2430>:  0x31    0x31    0x39    0x30    0x32    0x30    0x31    0x34
0x56559014 <z.2430+8>: 0x32    0x31    0x2d    0xe5    0xbc    0xa0    0xe7    0x91
0x5655901c <z.2430+16>: 0x9e    0x00

```

反汇编查看 x、y、z 在代码段的表示形式：

```

printf("%d\n",x);
1217:      8b 83 34 00 00 00      mov     0x34(%ebx),%eax
121d:      83 ec 08              sub     $0x8,%esp
1220:      50                  push    %eax
1221:      8d 83 34 e0 ff ff      lea     -0x1fcc(%ebx),%eax
1227:      50                  push    %eax
1228:      e8 53 fe ff ff        call    1080 <printf@plt>
122d:      83 c4 10              add     $0x10,%esp
printf("%f\n",y);
1230:      d9 45 f4              flds    -0xc(%ebp)
1233:      83 ec 04              sub     $0x4,%esp
1236:      8d 64 24 f8           lea     -0x8(%esp),%esp
123a:      dd 1c 24             fstpl   (%esp)
123d:      8d 83 38 e0 ff ff      lea     -0x1fc8(%ebx),%eax
1243:      50                  push    %eax
1244:      e8 37 fe ff ff        call    1080 <printf@plt>
1249:      83 c4 10              add     $0x10,%esp
printf("%s\n",z);
124c:      83 ec 0c              sub     $0xc,%esp
124f:      8d 83 38 00 00 00      lea     0x38(%ebx),%eax
1255:      50                  push    %eax
1256:      e8 35 fe ff ff        call    1090 <puts@plt>
125b:      83 c4 10              add     $0x10,%esp

```

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：在全局定义的数值型常量存储在数据段，在函数内定义的常量存储在栈中；全局变量和静态变量存储在数据段，局部变量存储在栈中。

字符串常量与变量在存储空间上的区别是：在全局定义的字符串常量存储在数据段，在函数内定义的常量存储在栈中；全局变量和静态变量存储在数据段，局部变量存储在栈中。

常量表达式在计算机中处理方法是：常量表达式在编译时直接被替换为立即数，得到计算结果。

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：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6     for(int i=0; i<argc; i++)
7     {
8         printf("argv[%d]:%s\n", i, argv[i]);
9     }
10    return 0;
11 }
```

```
zr@ubuntu: ~/code/codeblocks/mainpara
zr@ubuntu:~/code/codeblocks/mainpara$ gcc -m32 -g main.c -o main
zr@ubuntu:~/code/codeblocks/mainpara$ ./main x y z
argv[0]:./main
argv[1]:x
argv[2]:y
argv[3]:z
zr@ubuntu:~/code/codeblocks/mainpara$
```

代码及运行结果

```
(gdb) p &argv
$6 = (char ***) 0xffffcfff4
```

argv 的地址

```
(gdb) p argv
$1 = (char **) 0xffffd084
(gdb) p argv[0]
$2 = 0xffffd26f "/home/zr/code/codeblocks/mainpara/main"
(gdb) p argv[1]
$3 = 0xffffd296 "x"
(gdb) p argv[2]
$4 = 0xffffd298 "y"
(gdb) p argv[3]
$5 = 0xffffd29a "z"
```

argv 的内容

```
(gdb) p &argc
$7 = (int *) 0xffffcfff0
```

argc 的地址

```
(gdb) p &argv[1]  
$8 = (char **) 0xffffd088
```

x 的地址

```
(gdb) p &argv[2]  
$9 = (char **) 0xffffd08c
```

y 的地址

```
(gdb) p &argv[3]  
$10 = (char **) 0xffffd090
```

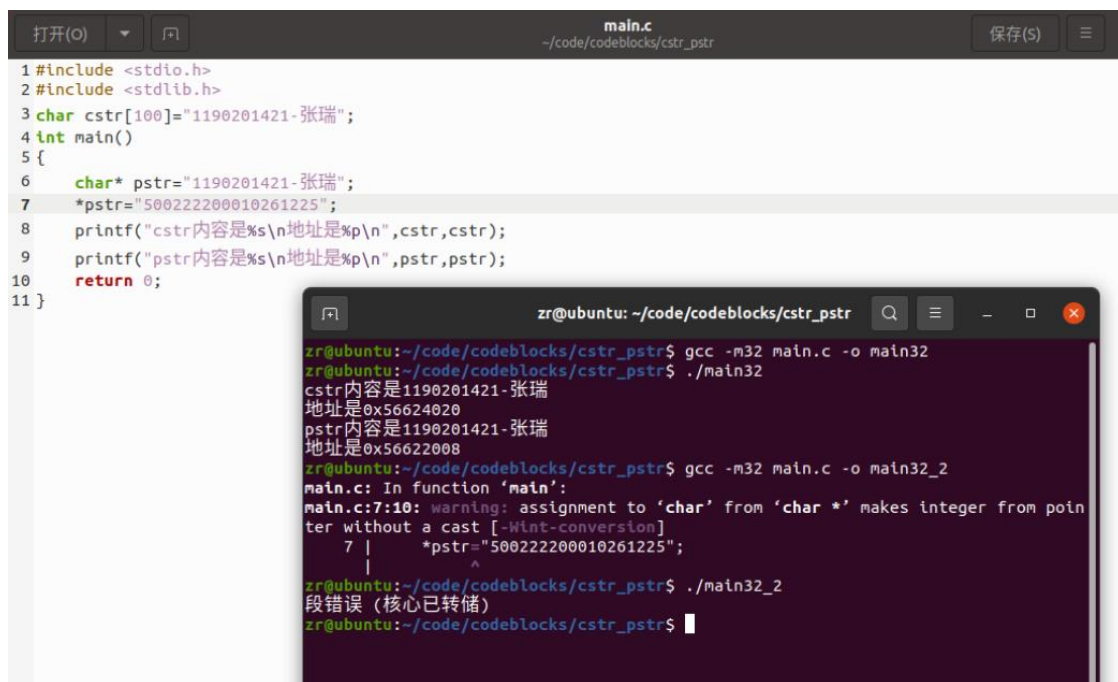
z 的地址

3.4 指针与字符串的区别

cstr 的地址与内容，pstr 的地址与内容截图：

```
main.c  
~/code/codeblocks/cstr_pstr  
保存(S)  
1 #include <stdio.h>  
2 #include <stdlib.h>  
3 char cstr[100]="1190201421-张瑞";  
4 int main()  
5 {  
6     char* pstr="1190201421-张瑞";  
7  
8     printf("cstr内容是%s\n地址是%p\n",cstr,cstr);  
9     printf("pstr内容是%s\n地址是%p\n",pstr,pstr);  
10    return 0;  
11 }  
  
zr@ubuntu: ~/code/codeblocks/cstr_pstr  
zr@ubuntu:~/code/codeblocks/cstr_pstr$ gcc -m32 main.c -o main32  
zr@ubuntu:~/code/codeblocks/cstr_pstr$ ./main32  
cstr内容是1190201421-张瑞  
地址是0x56624020  
pstr内容是1190201421-张瑞  
地址是0x56622008  
zr@ubuntu:~/code/codeblocks/cstr_pstr$
```

pstr 修改内容会出现什么问题_____会因访问了不该访问的段而出错（段错误）



The image shows a code editor window titled "main.c" with the following C code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 char cstr[100]="1190201421-张瑞";
4 int main()
5 {
6     char* pstr="1190201421-张瑞";
7     *pstr="500222200010261225";
8     printf("cstr内容是%s\n地址是%p\n",cstr,cstr);
9     printf("pstr内容是%s\n地址是%p\n",pstr,pstr);
10    return 0;
11 }
```

Below the code editor is a terminal window titled "zr@ubuntu: ~/code/codeblocks/cstr_pstr". It shows the compilation and execution of the code:

```
zr@ubuntu:~/code/codeblocks/cstr_pstr$ gcc -m32 main.c -o main32
zr@ubuntu:~/code/codeblocks/cstr_pstr$ ./main32
cstr内容是1190201421-张瑞
地址是0x56624020
pstr内容是1190201421-张瑞
地址是0x56622008
zr@ubuntu:~/code/codeblocks/cstr_pstr$ gcc -m32 main.c -o main32_2
main.c: In function 'main':
main.c:7:10: warning: assignment to 'char' from 'char*' makes integer from pointer without a cast [-Wint-conversion]
      7 |     *pstr="500222200010261225";
        |     ^
zr@ubuntu:~/code/codeblocks/cstr_pstr$ ./main32_2
段错误 (核心已转储)
zr@ubuntu:~/code/codeblocks/cstr_pstr$
```

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？

先用代码尝试比较了“计”、“算”、“机”、“系”、“统”几个字的排序：

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

```
int main()
```

```
{
```

```
    char a[]="计";
```

```
    char b[]="算";
```

```
    if(strcmp(a,b)<0)
```

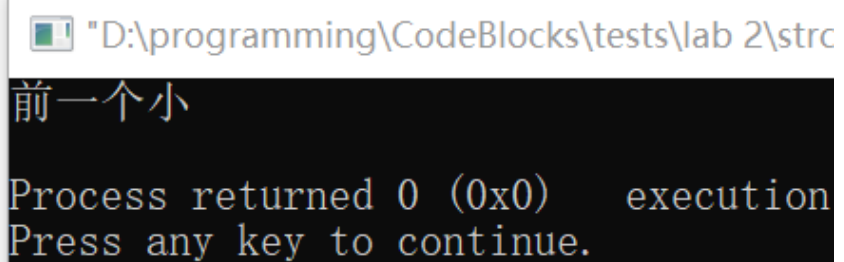
```
        printf("前一个小\n");
```

```
    else
```

```
        printf("后一个小\n");
```

```
    return 0;
```

```
}
```



```
"D:\programming\CodeBlocks\tests\lab 2\strc
前一个小

Process returned 0 (0x0) execution
Press any key to continue.
```

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

```
int main()
```

```
{
    char a[]="算";
    char b[]="机";
    if(strcmp(a,b)<0)
        printf("前一个小\n");
    else
        printf("后一个小\n");
    return 0;
}
```

"D:\programming\CodeBlocks\tests\lab 2\strc
后一个小
Process returned 0 (0x0) execution

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

```
int main()
```

```
{
    char a[]="机";
    char b[]="系";
    if(strcmp(a,b)<0)
        printf("前一个小\n");
    else
        printf("后一个小\n");
    return 0;
}
```

"D:\programming\CodeBlocks\tests\lab 2'
前一个小
Process returned 0 (0x0) execut
Press any key to continue.

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

```
int main()
```

```
{
    char a[]="系";
    char b[]="统";
    if(strcmp(a,b)<0)
        printf("前一个小\n");
    else
        printf("后一个小\n");
    return 0;
}
```

"D:\programming\CodeBlocks\tests\lab 2

后一个小

Process returned 0 (0x0) execut

Press any key to continue

再通过上网查阅，发现排序与 Unicode 不符，应该是按照 GBK 对汉字进行了排序：

字符	GBK编码10进制	GBK编码16进制 (GBK内码)	Unicode编码10进制	Unicode编码16进制
计	48326	BCC6	35745	8BA1
算	52195	CBE3	31639	7B97
机	48122	BBFA	26426	673A
系	53173	CFB5	31995	7CFB
统	52659	CDB3	32479	7EDF

查阅网址如下：http://www.mytju.com/classcode/tools/encode_gb2312.asp

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？

从上一问可以看出，排序的先后主要靠编码来实现。所以，要想实现按照姓氏笔画来排序的话，可以尝试建立一个姓氏及其编码的列表，里面各个姓氏有对应的按照笔画顺序排序的新构造的编码。当调用排序的操作时，将原本使用的编码转换为上述所说的新构造的编码，即可得到正确的排序结果。

第 5 章 数据变换与输入输出

5.1 提交 `cs_atoi.c`

详见压缩文件

5.2 提交 `cs_atof.c`

详见压缩文件

5.3 提交 `cs_itoa.c`

详见压缩文件

5.4 提交 `cs_ftoa.c`

详见压缩文件

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：

查阅《深入理解计算机系统第三版》P625~626 得知——应用程序是通过分别调用 `read` 和 `write` 函数来执行输入输出的。

如下所示：

```
#include <unistd.h>
```

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

//返回：若成功则为读的字节数，若 EOF 则为 0，若出错则为-1。

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

//返回：若成功则为写的字节数，若出错则为-1。

所以，OS 函数对于输入输出的数据有类型要求，操作是以字节为单位进行的。

第 6 章 整数表示与运算

6.1 提交 fib_dg.c

详见压缩文件

6.2 提交 fib_loop.c

详见压缩文件

6.3 fib 溢出验证

int 时从 n= 46 时溢出, long 时 n= 46 时溢出。

unsigned int 时从 n= 47 时溢出, unsigned long 时 n= 47 时溢出。

fib_dg 的运行结果:

 "D:\programming\CodeBlocks\tests\lab 2\fib_dg\bin\Debug\fib_dg.exe"

```
int的n为46
long的n为46
unsigned int的n为47
unsigned long的n为47
```

```
Process returned 0 (0x0)   execution time : 770.056 s
Press any key to continue.
```

fib_loop 的运行结果:

 "D:\programming\CodeBlocks\tests\lab 2\fib_loop\bin\Debug\fib_loop.exe"

```
int的n为46
long的n为46
unsigned int的n为47
unsigned long的n为47
```

```
Process returned 0 (0x0)   execution time : 0.271 s
Press any key to continue.
```

6.4 除以 0 验证

除以 0:

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

```
int main()
{
    int x=1;
    printf("%f\n",1/0);
    return 0;
}
```

"D:\programming\CodeBlocks\tests\lab 2\div\main.exe"

Process returned -1073741676 (0xC0000094) execution time : 1.135 s
Press any key to continue.

除以极小浮点数，截图：

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

```
int main()
{
    int x=1;
    printf("%f\n",1/0.0);
    return 0;
}
```

"D:\programming\CodeBlocks\tests\lab 2\div\main.exe"

1. #INF00

Process returned 0 (0x0) execution time : 0.207 s
Press any key 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。

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。

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）。

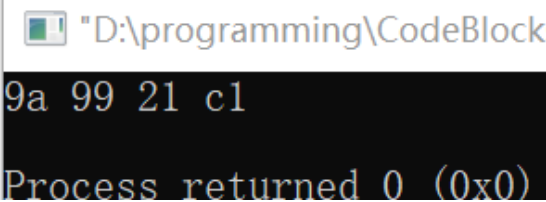
$-10.1 \Rightarrow S: 1$
 $(10.1)_{10} = (1010.0001100110011\dots)_2$
 $= 1.010001100110011\dots \times 2^3$
 $E=3 \quad \text{exp} = E + \text{Bias} = 3 + 127 = 130$
 由 $S \quad \text{exp} \quad \text{frac}$ 得表示为:
 $11000000101000011001100110011001\dots$
 注意左端在虚线处舍入, 最终为
 $\frac{1100}{c} \quad \frac{0001}{1} \quad \frac{0010}{2} \quad \frac{0001}{1} \quad \frac{1001}{9} \quad \frac{1001}{9} \quad \frac{1001}{9} \quad \frac{1010}{a}$
 小端:
 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");
}

int main()
{
    float x=-10.1;
    show_bytes((byte_pointer) &x, sizeof(float));
    return 0;
}
```



```
"D:\programming\CodeBlock
9a 99 21 c1
Process returned 0 (0x0)
```

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。

提交子程序 float0.c

```
int main()
{
    printf("%f\n", 0.1/0);
    printf("%f\n", 0.1/0.0);
    return 0;
}
```

```
"D:\programming\CodeBlocks\t
1. #INF00
1. #INF00

Process returned 0 (0x0)
Press any key to continue.
```

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= -1190201421 , 现 x= -1190201472

```
#include <stdio.h>
#include <stdlib.h>
int x=-1190201421;
int main()
{
    float y=10261225;
    static char z[]="1190201421-张瑞";
    printf("%d\n",x);
    printf("%f\n",y);
    printf("%s\n",z);
    x=(int)(float)x;
    printf("%d\n",x);
    return 0;
}
```

```
/home/zr/code/codeblocks/datalocation/main
-1190201421
10261225.000000
1190201421-张瑞
-1190201472

Process returned 0 (0x0)   execution time : 0.040 s
Press ENTER to continue.
```

7.5 讨论 1：有多少个 int 可以用 float 精确表示

有 150994943 个 int 数据可以用 float 精确表示。

是哪些数据呢？绝对值转换为二进制表达形式后，位数不超过 24 位的均可表示，即 0x00000000—0x00ffffff，共 2^{24} 个。超过 24 位但不超过 31 位部分数据，如 0x00800000—0x00ffffff，逐位左移直至第二位为 1 的过程中的数均可，共 $2^{23} \times 7$ 个数。+0 和 -0 都被计算会多出一个，故共计 $(2^{24} + 2^{23} \times 7) \times 2 - 1 = 150994943$ 个。

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！

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

int main()
{
    int x=0x0ffffff8; //按规格化数改写为二进制小数时，小数点后24位有效位，且第23位为1，第24位为1
    int y=0x0ffffffe; //按规格化数改写为二进制小数时，小数点后24位有效位，且第23位为0，第24位为1

    printf("%x\n",(int)(float)x);
    printf("%x\n",(int)(float)y);
    return 0;
}
```

```
"D:\programming\CodeBlocks\tests\lab 2\float\bi
10000000
fffffe0
```

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？3 个是哪些呢？0.25、0.5、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）： $(-2^{-125}, 2^{-125})$ 、 2^{-149} (正 0 负 0 算为两个)、 $(-2^{128}, -2^{127})$ 和 $(2^{127}, 2^{128})$ 、 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 2.02824×10^{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。

由前一问可以看出，浮点数可能存在不可区分的情况，故不可以用直接比较。

法 1：（查看浮点数对应的二进制表现形式）（不包含 NaN 的情况）

（1）先比较符号位，一般情况下，0 为正数，1 为负数，若符号位不同，则得出结果。特别地，应小心 +0 和 -0，两者符号位虽不同，但大小相等；

（2）符号位相同，再比较阶码段，同为正数情况下，阶码位大的数更大；反之，则更小；

（3）阶码段也相同，比较尾数段，同为正数情况下，尾数段大的数更大；反之，则更小；

（4）若全部相同，则认为在一定的误差范围内，两数相等（实际上并不一定相等）。

法 2：（浮点数相等时应用 $\text{fabs}(a-b) \leq 1e-6$ ）

如果 $\text{fabs}(a-b) \leq 1e-6$ ，则 ab 相等；

若 $\text{fabs}(a-b) > 1e-6 \&\& a < b$ ，则 $a < b$ ；

若 $\text{fabs}(a-b) > 1e-6 \&\& a > b$ ，则 $a > b$ 。

第 8 章 舍位平衡的讨论

8.1 描述可能出现的问题

在报表的数据统计中，常常会根据精度呈现或者单位换算等要求，需要对数据执行四舍五入的操作，这种操作称为舍位处理。简单直接的舍位处理有可能会带来隐患，原本平衡的数据关系可能会被打破。例如，保留一位小数的原始的数据是 $4.5+4.5=9.0$ ，而四舍五入只保留整数部分后，平衡关系就变为 $5+5=9$ 了，看上去明显是荒谬的。

为了保证报表中数据关系的正确，就需要调整舍位之后的数据，使得数据重新变得平衡，这样的调整就叫做舍位平衡。

8.2 给出完美的解决方案

舍位后总计产生的误差，称为“平衡差”，舍位平衡其实就是消除平衡差的过程。

1. 单向舍位平衡

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。

Index	Member	
1	1.48	
2	0	
3	1.42	
4	0.32	
5	6.48	
6	0.98	Value
7	1.39	12.07

原始数据及其总和

Index	Member		
1	1.0		
2	0		
3	1.0		
4	0.0		
5	6.0		
6	1.0	Value	Value
7	1.0	12.0	10.0

舍入后数据、原始总和、舍入后数据的和

显然，此时出现了平衡差为 2，下面进行舍位平衡的处理：（3 个方法）

（1）直接把平衡差加到第一个数据上。

Index	Member	
1	3.0	
2	0	
3	1.0	
4	0.0	
5	6.0	
6	1.0	Value
7	1.0	12.0

这样虽然简单且处理了平衡差的问题，但使得数据显得较为不合理。

(2) 将平衡差按照“最小调整值”(即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)，对绝对值比较大的数据进行分担调整。

Index	Member	
1	2.0	
2	0	
3	1.0	
4	0.0	
5	7.0	
6	1.0	Value
7	1.0	12.0

这样能避免(1)中出现的问题，但需要对数据排序，时间成本较大。

(3) 将平衡差按照最小调整值，由不为0的数据依次分担。

Index	Member	
1	2.0	
2	0	
3	2.0	
4	0.0	
5	6.0	
6	1.0	Value
7	1.0	12.0

避免了排序操作，效率较高，常用。

2.双向舍位平衡

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，此时处理舍位平衡时，不仅要求最终的总计值准确，同时行向和列向计算的合计值也要与对应行、列的数据平衡。(5个步骤)

(1) 横向与纵向的非合计平衡差符号相同时，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。

(2) 同向的2个非合计平衡差符号相反时，只需要任选一行平衡差为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。

(3) 某个合计平衡差与另一方向的非合计平衡差符号相反时，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。

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

(5) 两个方向合计平衡差的符号相同时，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

通过在 Windows 和 Linux 下配置 32 位/64 位环境查看数据类型的大小和位置，我更加熟悉了 CB 和 Linux 系统，掌握了终端下 gdb、objdump 等指令的使用方法，并初次尝试查看反汇编，了解到了数值型常量和变量，字符串常量和变量在存储空间上的不同。

通过自主编写 cs_atoi/cs_atof 字符串转正数/浮点数和 cs_itoa/cs_ftoa 正数/浮点数转字符串函数，体会到了不同数据类型在存储和转换时的区别。

通过验证斐波那契数列的溢出情况，再次体会到了 int/long/unsigned int/unsigned long 数据类型在 32 位系统中所能表示的范围有限，需注意数据的溢出。并验证了 C 语言整数和浮点数各自除以 0 和极小浮点数的情况，再次体会溢出。

通过浮点数据的表示和运算，更加深刻理解到了 IEEE754 编码方式，以及 int 和 float 数据在类型转换时精度丢失的问题。

通过对 float 的微观与宏观世界的探索，更加深刻体会到了 float 数据“越靠近原点越稠密，越远离原点越稀疏”的分布特点以及其表示范围和精度的有限性。

最后一部分的舍位平衡，让我在网上查阅了大量资料，锻炼了我的资料搜集能力和自学理解能力，启发了我对于这类从未设想过的问题的思考。

9.2 请给出对本次实验内容的建议

实验任务多而杂，且难度较高（好像还有点超前），实验指导书的翻译也是模模糊糊，很多地方都不懂在说什么，老师上课也没怎么讲授实验中的重难点，整个实验全靠自己一点点硬啃，查了很多资料，费了很多时间，压力偏大。

建议修改实验指导书，使其更为丰富详实，甚至可以对实验内容进行一些删改。希望老师在实验课上也能多讲授一些有助于完成实验的重难点操作。

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] 谌颖. 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>.
- [7] https://blog.csdn.net/sanganqi_wusuierzi/article/details/54783958
- [8] <https://blog.csdn.net/wwchao2012/article/details/79980514>
- [9] https://blog.csdn.net/weixin_33726318/article/details/89616837