哈爾濱工業大學

实验报告

实 验（二）

题　　目 DataLab 数据表示

专　　业 计算机类

学　　号 1190200523

班　　级 1903002

学　　生 石翔宇

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.4.2

计算机科学与技术学院

目 录

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

- Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- 16GB RAM
- 1TB HDD + 512G SSD

1.2.2 软件环境

- Windows 10 21H1
- Ubuntu 20.04 LTS

1.2.3 开发工具

- VSCode，CodeBlocks，gcc+gdb

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
  - 先用递归程序实现，会出现什么问题？

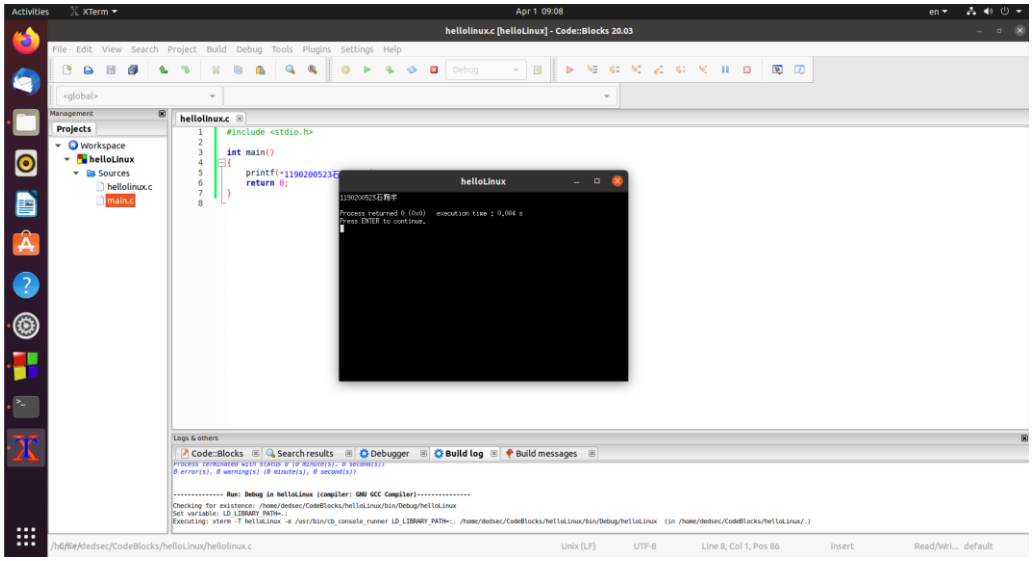
- 再用循环方式实现。

- 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-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

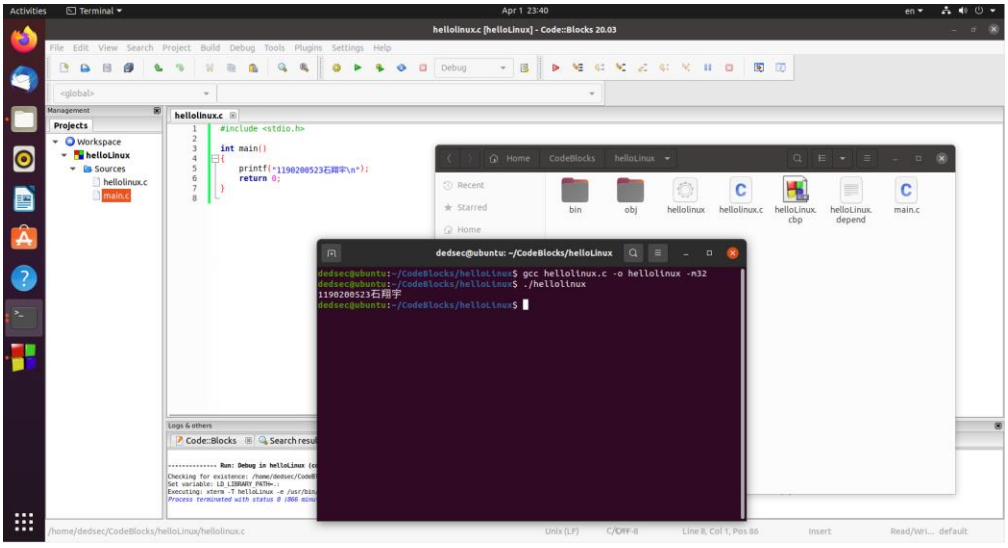
2.1 Ubuntu 下 CodeBlocks 安装

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

C 编译器对 sizeof 的实现方式：sizeof 是一个操作符，由编译器来计算，编译阶段计算出结果，在运行时是个常量。

3.2 数据的位置-地址

打印 x、y、z 输出的值：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ gcc test_addr.c -o test_addr
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ ./test_addr
x = -1190200523
y = 131182200003105536.000000
z = 1190200523-石翔宇
dedsec@ubuntu:~/CSAPP/Lab/Lab2$
```

反汇编查看 x、y、z 的地址，每字节的内容：

```
(gdb) p &x
$2 = (const int *) 0x2004 <x>
(gdb) p /x x
$3 = 0xb90efb35
```

x 的地址为 0x2004，内容为 0xb90efb35

```
11          printf("y = %lf = 0x%x\n", y, *yp);
(gdb) n
y = 131182199734009856.000000 = 0x5be906c2
12          printf("z = %s\n", z);
(gdb) p &y
$1 = (float *) 0x7fffffffdf6c
```

y 的地址为 0x7fffffffdf6c，内容为 0x5be906c2

```
(gdb) p &z
$2 = (char (*)[100]) 0x555555558020 <z>
(gdb) p /x z
$3 = {0x31, 0x31, 0x39, 0x30, 0x32, 0x30, 0x30, 0x35, 0x32, 0x33, 0x2d, 0xe7, 0x9f, 0xb3,
  0xe7, 0xbf, 0x94, 0xe5, 0xae, 0x87, 0x0 <repeats 80 times>}
```

z 的地址为 0x555555558020，内容为{0x31, 0x31, 0x39, 0x30, 0x32, 0x30, 0x30, 0x35, 0x32, 0x33, 0x2d, 0xe7, 0x9f, 0xb3, 0xe7, 0xbf, 0x94, 0xe5, 0xae, 0x87, 0x0 <repeats 80 times>}

反汇编查看 x、y、z 在代码段的表示形式。

```
Disassembly of section .rodata:

0000000000000000 <x>:
   0:   35 fb 0e b9 78          xor    $0x78b90efb,%eax
   5:   20 3d 20 25 64 0a       and    %bh,0xa642520(%rip)        # a64252b <x+0xa64252b>
   b:   00 79 20                add    %bh,0x20(%rcx)
   e:   3d 20 25 6c 66          cmp    $0x666c2520,%eax
  13:   20 3d 20 30 78 25       and    %bh,0x25783020(%rip)        # 25783039 <x+0x25783039>
  19:   78 0a                   js     25 <x+0x25>
  1b:   00 7a 20                add    %bh,0x20(%rdx)
  1e:   3d 20 25 73 0a          cmp    $0xa732520,%eax
  23:   00 c2                   add    %al,%dl
  25:   06                      (bad)
  26:   e9                      .byte 0xe9
  27:   5b                      pop    %rbx
```

x 在代码段中表示如图红框所示

```
Disassembly of section .rodata:

0000000000000000 <x>:
   0:   35 fb 0e b9 78          xor    $0x78b90efb,%eax
   5:   20 3d 20 25 64 0a       and    %bh,0xa642520(%rip)        # a64252b <x+0xa64252b>
   b:   00 79 20                add    %bh,0x20(%rcx)
   e:   3d 20 25 6c 66          cmp    $0x666c2520,%eax
  13:   20 3d 20 30 78 25       and    %bh,0x25783020(%rip)        # 25783039 <x+0x25783039>
  19:   78 0a                   js     25 <x+0x25>
  1b:   00 7a 20                add    %bh,0x20(%rdx)
  1e:   3d 20 25 73 0a          cmp    $0xa732520,%eax
  23:   00 c2                   add    %al,%dl
  25:   06                      (bad)
  26:   e9                      .byte 0xe9
  27:   5b                      pop    %rbx
```

y 在代码段中表示如图红框所示

```
Disassembly of section .data:

0000000000000000 <z.2317>:
   0:   31 31                   xor    %esi,(%rcx)
   2:   39 30                   cmp    %esi,(%rax)
   4:   32 30                   xor    (%rax),%dh
   6:   30 35 32 33 2d e7       xor    %dh,-0x18d2ccce(%rip)        # ffffffffe72d333e <z.2317+0
xffffffffe72d333e>
   c:   9f                      lahf
   d:   b3 e7                   mov    $0xe7,%bl
   f:   bf 94 e5 ae 87          mov    $0x87aee594,%edi
        ...
```

z 在代码段中表示如图红框所示

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：数值型常量一般存储在常量存储区（.rodata 段），而变量则存储在.data 段

字符串常量与变量在存储空间上的区别是：字符串变量的名字及其所需的存储空间是显式定义的，并通过名字来引用相应的字符串变量，存储在.data 段。而字符串常量所需的存储空间是隐式定义的，并且没有名字，存储在.rodata 段。

常量表达式在计算机中处理方法是：尽量用左右移代替乘除法，编译阶段就计算出了常量表达式的值，储存在内存中

3.2 提示：

①在 linux 下生成可执行程序，假设是 a.out。然后用 objdump -dx a.out ＞a-dump.s 生成反汇编文件 a-dump.s，查看 a-dump.s

②gdb ./a.out ☑ layout asm ☑ b main ☑ r ☑ disp argc ☑….

3.3 main 的参数分析

反汇编查看 x、y、z 的地址截图 4；

命令行传递参数，反汇编观察 argc、argv 的地址与内容，截图 4。

```
(gdb) p argv[1]
$17 = 0x7fffffffe37a "-1190200523"
(gdb) p &argv[1]
$18 = (char **) 0x7fffffffe030
(gdb) p argv[2]
$19 = 0x7fffffffe386 "131182200003105535"
(gdb) p &argv[2]
$20 = (char **) 0x7fffffffe038
(gdb) p argv[3]
$21 = 0x7fffffffe399 "1190200523-石翔宇"
(gdb) p &argv[3]
$22 = (char **) 0x7fffffffe040
(gdb) p argc
$23 = 4
(gdb) p &argc
$24 = (int *) 0x7fffffffdf2c
```

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

```
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ objdump -D test_str > test_str_dump.s
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ ./test_str
content: 1190200523-石翔宇 address: 0x270c8020
content: 1190200523-石翔宇 address: 0x270c6004
```

pstr 修改内容会出现什么问题：访问无效内存，出现段错误

```
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ gcc test_str.c -o test_str
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ ./test_str
Segmentation fault (core dumped)
```

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

答：strcmp 按照汉字的编码（例 UTF-8）的二进制从小到大对汉字排序，直到出现不同的字符或遇到“\0”为止。

4.3 讨论：按照姓氏笔画排序的方法实现（选做，不做要求）

分析论述：应该怎么实现呢？

答：提前预处理出一个按照姓氏笔画排序的汉字列表，按照该列表得出先后顺序，即可根据姓氏笔画排序。

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：有。

OS 的函数将输入输出的数据都看成字符串来处理。对于输入，OS 先接收输入的字符串，再进行进一步处理；对于输出，将数据转换成字符串再输出。

例如：

sprintf 函数：输出 int 类型，输入要求为字符串地址。

atoi 函数：输出 int 类型，输入字符串首地址。

atof 函数：输出 double 类型，输入字符串首地址。

itoa 函数：返回转换后字符串首地址，输入待转换 int 数以及转换进制。

ftoa 函数：返回转换后字符串首地址，输入 float 和基数。

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：截图 1

```
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ gcc div0.c -o div0
div0.c: In function ‘main’:
div0.c:6:28: warning: division by zero [-Wdiv-by-zero]
    6 |     printf("1 / 0 = %d", x / 0);
      |                           ^
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ ./div0
Floating point exception (core dumped)
```

除以极小浮点数，截图：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ gcc div0.c -o div0
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ ./div0
1 / 1e-30 = 99999999999999987914713648332 8.000000
```

第 7 章 浮点数据的表示与运算

7.1 正数表示范围

写出 float/double 类型最小的正数、最大的正数（非无穷）

Float:

$$(0\ 00000000\ 00000000000000000000001)_2 = (2^{-23} * 2^{-126})_{10}$$
$$= 1.401298464324817 * 10^{-45}$$
$$(011111110\ 11111111111111111111111)_2 = \left((2-2^{-23}) * 2^{127}\right)_{10}$$
$$= 3.4028234663852886 * 10^{38}$$

Double:

$$(0\ 00000000000\ 0000000000000000000000000000000000000000000000000001)_2$$
$$= (2^{-52} * 2^{-1022})_{10} = 5 * 10^{-324}$$
$$(0\ 11111111110\ 1111111111111111111111111111111111111111111111111111)_2$$
$$= \left((2-2^{-52}) * 2^{1023}\right)_{10} = 1.7976931348623157 * 10^{308}$$

7.2 浮点数的编码计算

（1）按步骤写出 float 数-1.1 的浮点编码计算过程，写出该编码在内存中从低地址字节到高地址字节的 16 进制数值

1. 十进制转二进制：$-1.0001100110011[0011]$
2. 变成：$-(1 + 0.0001100110011[0011]) * 2^0$
3. 即：$-(1 + 0.0001100110011[0011]) * 2^{127-127}$
4. 变成二进制：1 01111111 00011001100110011001100**11**
5. 向偶数舍入：1 01111111 00011001100110011001101
6. 分位：1011 1111 1000 1100 1100 1100 1100 1101
7. 转成 16 进制：bf8ccccd

（2）验证：编写程序，输出值为-1.1 的浮点变量其各内存单元的数值，截图。

```
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ gcc test_float.c -o test_float ; ./test_float
test_float.c: In function ‘main’:
test_float.c:7:20: warning: initialization of ‘unsigned int *’ from incompatible pointer type ‘float *’ [-Wincompatible-pointer-types]
    7 |     unsigned *ap = &a;
      |                    ^
test_float.c:8:26: warning: initialization of ‘unsigned char *’ from incompatible pointer type ‘float *’ [-Wincompatible-pointer-types]
    8 |     unsigned char *app = &a;
      |                          ^
-1.100000 = 0xbf8ccccd
```

7.3 特殊浮点数值的编码

（1）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。

```
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ ./floatx
0.000000        0x00000000
-0.000000       0x80000000
0.000000        0x00000001
340282346638528859811704183484516925440.000000  0x7f7fffff
0.000000        0x00800000
inf     0x7f800000
nan     0x7f800001
```

（2）提交子程序 floatx.c

7.4 浮点数除 0

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

```
dedsec@ubuntu:~/CSAPP/Lab/Lab2$ gcc float0.c -o float0 ; ./float0
float0.c: In function ‘main’:
float0.c:6:30: warning: division by zero [-Wdiv-by-zero]
    6 |     printf("x / 0 = %f\n", x / 0);
      |                              ^
x / 0 = inf
x / 1e-100 = 1000000000000000015902891109759918046836080856394528138978132755774783877217038106081346998585681510 4.000000
```

（2）提交子程序 float0.c

7.5 Float 的微观与宏观世界

按照阶码的数值区域，float 编码最密集区域的阶码编码是：00000000 和 00000001，该区域中相邻浮点数编码的数值的间距是：$2^{-23} * 2^{-126}$；float 编码最稀疏区域的阶码编码是：11111110，该区域中相邻浮点数编码的数值的间距是：$2^{-23} * 2^{127}$。

最小正数变成十进制科学记数法，最可能能精确到多少 1.401298464324817e-45

最大正数变成十进制科学记数法，最可能能精确到多少 3.4028234663852886e+38

7.6 讨论：任意两个浮点数的大小比较

论述比较方法以及原因。

方法：将两个浮点数a,b相减得到差c（$c = a - b$），将c取绝对值得到d。取一极小数ϵ与d相比较，若d小于ϵ，则a与b相等。若d大于ϵ，则判断c是否大于0（c 的符号位是 0 还是 1）。若c大于0，则$a > b$；若c小于0，则$a < b$。

原因：浮点数的存储有精度限制，有些数无法用浮点数精确表示，因此无法直接对浮点数进行大小比较。

第 8 章 舍尾平衡的讨论

8.1 描述可能出现的问题

在对数据进行统计的过程中，常常由于精度等问题对数据进行舍入。若舍入方法不合理，在大量的数据积累后，可能会产生很大的误差。

8.2 给出完美的解决方案

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。
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，这种情况下处理舍位平衡时就复杂得多了。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，这种情况下的舍位平衡称为双向舍位平衡。

第 9 章 总结

9.1 请总结本次实验的收获

- 对数据的存储有了更深的理解。
- 学习了 UTF-8 编码的编码方式。
- 深入理解了浮点数的表示方法，也对浮点数的处理有了更清楚的认识。

9.2 请给出对本次实验内容的建议

- 希望实验有关的材料排版能够更加严谨易读，去除冗杂部分，强调重点部分，精简实验内容。
- 部分材料模糊不清，有歧义，希望可以修改。

注：本章为酌情加分项。

参考文献

[1] 大卫 R.奥哈拉伦，兰德尔 E。布莱恩特. 深入理解计算机系统[M]. 机械工业出版社.2017.7