

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

班 级

学 生 Youngsc

指 导 教 师 郑贵滨

实 验 地 点 G712

实 验 日 期 2020-4-2

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	- 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)	- 9 -
3.4 指针与字符串的区别 (2 分)	- 10 -
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_ATOI.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 -
第 7 章 浮点数据的表示与运算	- 16 -
7.1 正数表示范围	- 16 -
7.2 浮点数的编码计算	- 16 -

7.3 特殊浮点数值编码	- 16 -
7.4 浮点数除 0	- 17 -
7.5 FLOAT 的微观与宏观世界	- 17 -
7.6 讨论：任意两个浮点数的大小比较	- 18 -
第 8 章 舍尾平衡的讨论	- 19 -
8.1 描述可能出现的问题	- 19 -
8.2 给出完美的解决方案	- 19 -
第 9 章 总结	- 20 -
9.1 请总结本次实验的收获	- 20 -
9.2 请给出对本次实验内容的建议	- 20 -
参考文献	- 21 -

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

Legion Y7000P 2019 PG0
CPU: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz (12 CPUs), ~2.6GHz
RAM: 16384MB

1.2.2 软件环境

Windows 10 家庭中文版 64-bit
Ubuntu 20.04.2 LTS
VMware® Workstation 16 Player 16.1.0 build-17198959

1.2.3 开发工具

Microsoft Visual Studio Community 2019 版本 16.9.2
Microsoft Visual 1.54.3
GCC 9.3.0

1.3 实验预习

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，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

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

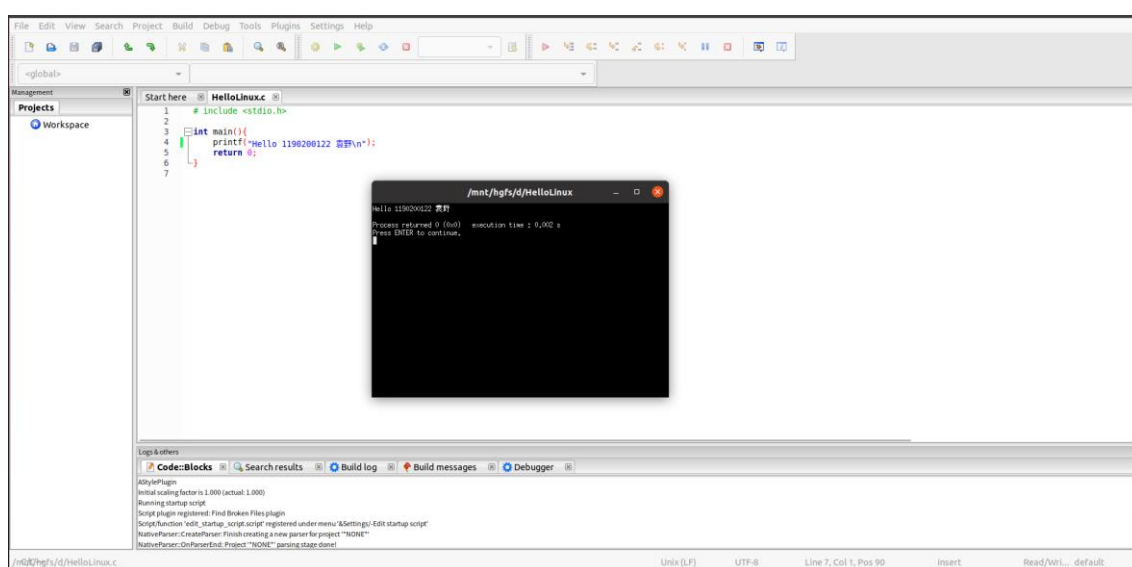


图 2-1 Ubuntu 下 CodeBlocks 截图

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。

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc HelloLinux.c -o a -m32
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./a
Hello 1190200122 袁野
yuanye@1190200122-yuanye:/mnt/hgfs/d$
```

图 2-2 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

3.1 提示：在 linux 和 windows 两种平台下，用两种编译器观察。不限定非得是 Codeblock

C 编译器对 sizeof 的实现方式：编译器预处理出类型大小

3.2 数据的位置-地址

打印 x、y、z 输出的值：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./xyz
-1190200122
130727199488606208.000000
1190200122-袁野
```

截图一

反汇编查看 x、y、z 的地址，每字节的内容：

```
(gdb) x/4bx &x
0x56559008 <x>: 0xc6 0xfc 0x0e 0xb9
(gdb) x/4bx &y
0xffffcf3c: 0xd9 0x37 0xe8 0x5b
(gdb) x/8bx &z
0x5655900c <z.1912>: 0x12 0x70 0x55 0x56 0x00 0x00 0x00 0x00
```

截图二

x 地址：0x56559008 内容：0xc6 0xfc 0x0e 0xb9
y 地址：0xffffcf3c 内容：0xd9 0x37 0xe8 0x5b

z 地址: 0x5655900c 内容 0x0e 0x60 0x55 0x55 0x55 0x55
0x00 0x00

反汇编查看 x、y、z 在代码段的表示形式。

```

179 000011cd <main>:
180      11cd:      f3 0f 1e fb      endbr32
181      11d1:      8d 4c 24 04      lea     0x4(%esp),%ecx
182      11d5:      83 e4 f0         and     $0xffffffff0,%esp
183      11d8:      ff 71 fc         pushl   -0x4(%ecx)
184      11db:      55              push    %ebp
185      11dc:      89 e5           mov     %esp,%ebp
186      11de:      53              push    %ebx
187      11df:      51              push    %ecx
188      11e0:      e8 eb fe ff ff   call    10d0 <__x86.get_pc_thunk.bx>
189      11e5:      81 c3 f3 2d 00 00 add     $0x2df3,%ebx
190      11eb:      83 ec 08         sub     $0x8,%esp
191      11ee:      8d 83 30 e0 ff ff lea     -0x1fd0(%ebx),%eax
192      11f4:      50              push    %eax
193      11f5:      68 fb 06 7d 43   push    $0x437d06fb
194      11fa:      68 00 00 00 20   push    $0x20000000
195      11ff:      ff b3 30 00 00 00 pushl   0x30(%ebx)
196      1205:      8d 83 42 e0 ff ff lea     -0x1fbe(%ebx),%eax
197      120b:      50              push    %eax
198      120c:      6a 01           push    $0x1
199      120e:      e8 6d fe ff ff   call    1080 <__printf_chk@plt>
200      1213:      83 c4 20         add     $0x20,%esp
201      1216:      b8 00 00 00 00   mov     $0x0,%eax
202      121b:      8d 65 f8         lea     -0x8(%ebp),%esp
203      121e:      59              pop     %ecx
204      121f:      5b              pop     %ebx
205      1220:      5d              pop     %ebp
206      1221:      8d 61 fc         lea     -0x4(%ecx),%esp
207      1224:      c3              ret
208      1225:      66 90           xchg    %ax,%ax
209      1227:      66 90           xchg    %ax,%ax
210      1229:      66 90           xchg    %ax,%ax
211      122b:      66 90           xchg    %ax,%ax
212      122d:      66 90           xchg    %ax,%ax
213      122f:      90              nop
214

```

截图三

x 与 y 在__汇编语言代码翻译成目标机器指令__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: __宏常量不进行储存, const 常量储存在代码段, 全局变量和静态变量处于存在数据段, 局部变量储存在堆栈段。__

字符串常量与变量在存储空间上的区别是: __字符串常量保存在代码段, 静态变量和全局变量保存在数据段, 局部变量在堆栈段__

常量表达式在计算机中处理方法是: __在编译时进行计算求出该值。__

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;

```
(gdb) p *(argv+1)
$1 = 0xffffd1ed "x"
(gdb) p *(argv+2)
$2 = 0xffffd1ef "y"
(gdb) p *(argv+3)
$3 = 0xffffd1f1 "z"
```

x、y、z 的地址

命令行传递参数，反汇编观察 argc、argv 的地址与内容，截图 4。

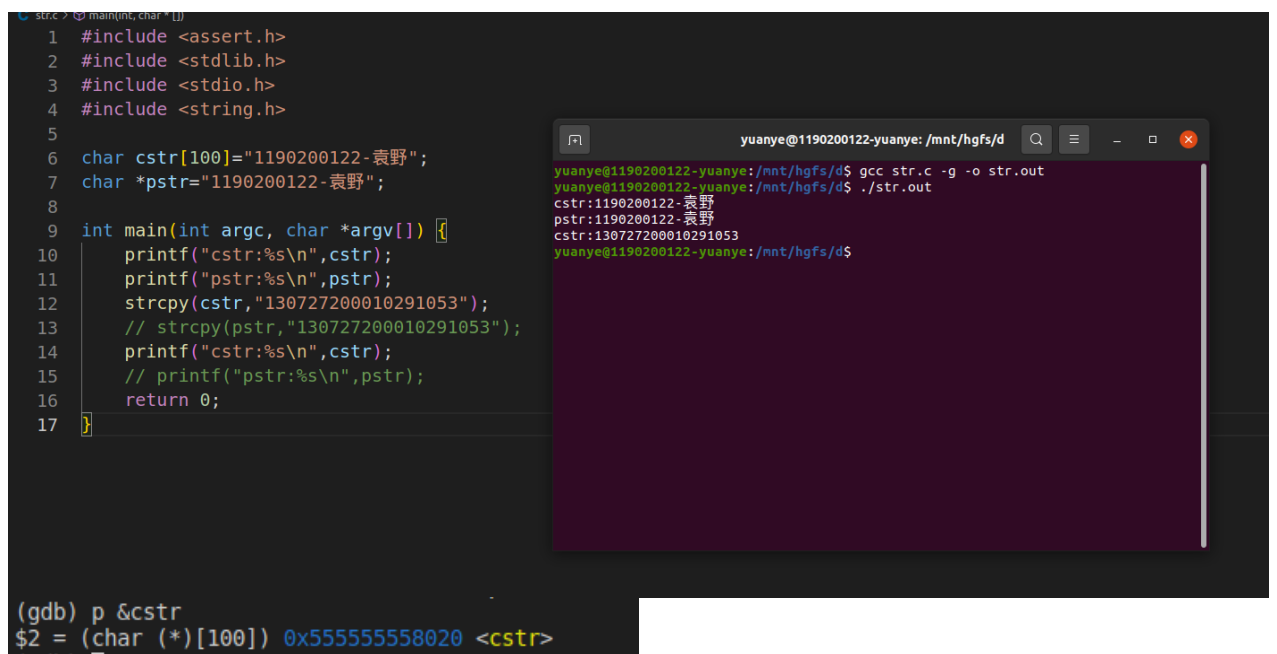
```
(gdb) p argc
$4 = 4
(gdb) p &argc
$5 = (int *) 0xffffcf50
```

argc 的内容和地址

```
(gdb) p argv
$8 = (char **) 0xffffcfe4
(gdb) p &argv
$9 = (char ***) 0xffffcf54
```

argv 的内容和地址

3.4 指针与字符串的区别



```
1 #include <assert.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <string.h>
5
6 char cstr[100]="1190200122-袁野";
7 char *pstr="1190200122-袁野";
8
9 int main(int argc, char *argv[]) {
10     printf("cstr:%s\n",cstr);
11     printf("pstr:%s\n",pstr);
12     strcpy(cstr,"130727200010291053");
13     // strcpy(pstr,"130727200010291053");
14     printf("cstr:%s\n",cstr);
15     // printf("pstr:%s\n",pstr);
16     return 0;
17 }
```

```
yuanye@1190200122-yuanye: /mnt/hgfs/d$ gcc str.c -g -o str.out
yuanye@1190200122-yuanye: /mnt/hgfs/d$ ./str.out
cstr:1190200122-袁野
pstr:1190200122-袁野
cstr:130727200010291053
yuanye@1190200122-yuanye: /mnt/hgfs/d$
```

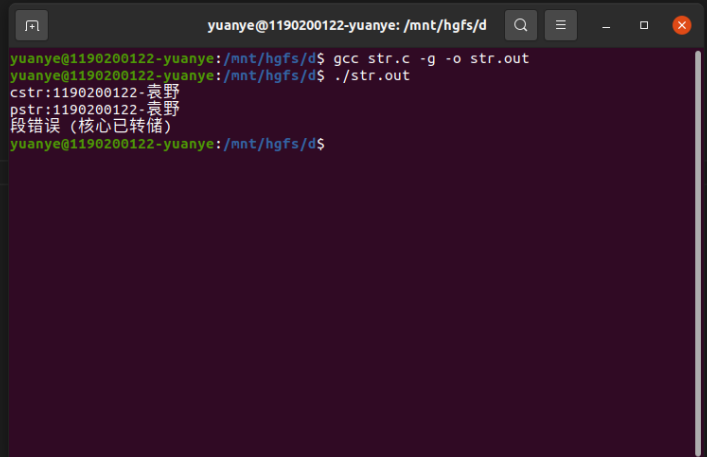
```
(gdb) p &cstr
$2 = (char (*)[100]) 0x55555558020 <cstr>
```

cstr 的地址与内容截图

```
1 #include <assert.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <string.h>
5
6 char cstr[100]="1190200122-袁野";
7 char *pstr="1190200122-袁野";
8
9 int main(int argc, char *argv[]) {
10     printf("cstr:%s\n",cstr);
11     printf("pstr:%s\n",pstr);
12     // strcpy(cstr,"130727200010291053");
13     strcpy(pstr,"130727200010291053");
14     // printf("cstr:%s\n",cstr);
15     printf("pstr:%s\n",pstr);
16     return 0;
17 }
```

```
(gdb) p &pstr
$1 = (char **) 0x555555558088 <pstr>
```

pstr 的内容与截图 截图 5



```
yuanye@1190200122-yuanye: /mnt/hgfs/d$ gcc str.c -g -o str.out
yuanye@1190200122-yuanye: /mnt/hgfs/d$ ./str.out
cstr:1190200122-袁野
pstr:1190200122-袁野
段错误 (核心已转储)
yuanye@1190200122-yuanye: /mnt/hgfs/d$
```

pstr 修改内容会出现什么问题____字符串指针为常量，不可修改，修改会导致段错误____

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

strcmp 是根据变量储存的数据的二进制大小来进行比较的，之所以能够按照字典序来排序英文，是因为 ascii 是按照字典序来分配的。

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？

使用一套按笔画排序进行编码的编码系统；或者先统计出每个汉字的笔画顺序，每次比较时按照数据库中的笔画对于汉字进行排序。

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：有，OS 的函数的输入输出的数据为字符串处理。

第 6 章 整数表示与运算

6.1 提交 fib_dg.c

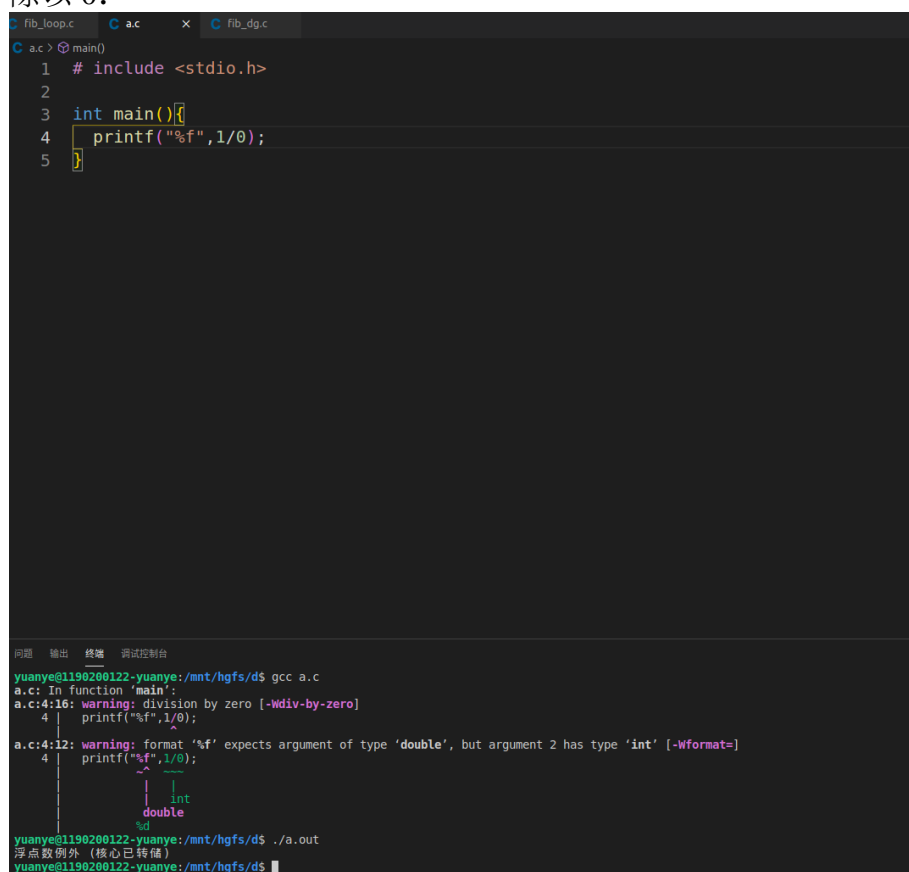
6.2 提交 fib_loop.c

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:



```
fib_loop.c  a.c  fib_dg.c
a.c > main()
1  #include <stdio.h>
2
3  int main(){
4      printf("%f",1/0);
5  }
```

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc a.c
a.c: In function 'main':
a.c:4:16: warning: division by zero [-Wdiv-by-zero]
4 | printf("%f",1/0);
  |                ^
a.c:4:12: warning: format '%f' expects argument of type 'double', but argument 2 has type 'int' [-Wformat=]
4 | printf("%f",1/0);
  |                ^~
  |                |
  |                int
  |                double
  |                %d
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./a.out
浮点数例外 (核心已转储)
yuanye@1190200122-yuanye:/mnt/hgfs/d$
```

截图 1

除以极小浮点数, 截图:

```
a.c > main()
1 #include <stdio.h>
2 #include <float.h>
3
4 int main(){
5     printf("%f\n", 1/1.3333e-38);
6 }
```

问题 输出 终端 调试控制台

```
75001875046876170384455462417032282112.000000yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc a.c
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./a.out
75001875046876170384455462417032282112.000000
yuanye@1190200122-yuanye:/mnt/hgfs/d$
```

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 最小为 $1.4e-45$ 最大为 $3.4028235e38$

double 最小为 $4.9e-324$ 最大为 $1.7976931348623157e308$

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

$-1.1 = 0b-1.000110011\cdots$

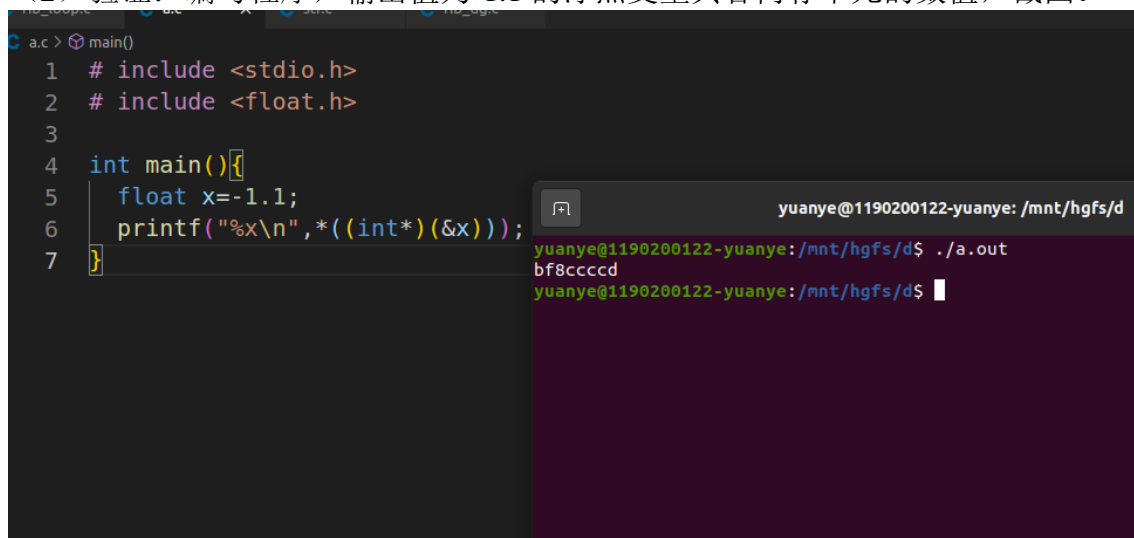
尾数为 $0b1.00011001100110011001101$

阶码为 0, e 为 $127 = 0b10000000$

754 格式为 $0x\ 1011\ 1111\ 1000\ 1100\ 1100\ 1100\ 1101$

十六进制即为 BF 8C CC CD

（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。



```
a.c > main()
1 #include <stdio.h>
2 #include <float.h>
3
4 int main(){
5     float x=-1.1;
6     printf("%x\n",*((int*)&x));
7 }
```

```
yuanye@1190200122-yuanye: /mnt/hgfs/d
yuanye@1190200122-yuanye: /mnt/hgfs/d$ ./a.out
bf8ccccd
yuanye@1190200122-yuanye: /mnt/hgfs/d$
```

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。

```

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

3
4 int main() {
5     float pos_zero = +0.0f;
6     float neg_zero = -0.0f;
7     float pos_max, pos_min, pos_norm_min, pos_inf, nan;
8     *(int*)&pos_max = 0x7F7FFFFF;
9     *(int*)&pos_min = 0x00000001;
10    *(int*)&pos_norm_min = 0x00800000;
11    *(int*)&pos_inf = 0x7F800000;
12    *(int*)&nan = 0x7F800001;
13    printf("pos_zero = %.9g hex: %a\n", pos_zero, pos_zero);
14    printf("neg_zero = %.9g hex: %a\n", neg_zero, neg_zero);
15    printf("pos_max = %.9g hex: %a\n", pos_max, pos_max);
16    printf("pos_min = %.9g hex: %a\n", pos_min, pos_min);
17    printf("pos_norm_min = %.9g hex: %a\n", pos_norm_min, pos_norm_min);
18    printf("pos_inf = %.9g hex: %a\n", pos_inf, pos_inf);
19    printf("NAN = %.9g hex: %a\n", nan, nan);
20    return 0;
21 }

```

```

yuanyue@1190200122-yuanyue: /mnt/hgfs/d$ gcc floatx.c
yuanyue@1190200122-yuanyue: /mnt/hgfs/d$ ./a.out
pos_zero = 0 hex: 0x0p+0
neg_zero = -0 hex: -0x0p+0
pos_max = 3.40282347e+38 hex: 0x1.fffffep+127
pos_min = 1.40129846e-45 hex: 0x1p-149
pos_norm_min = 1.17549435e-38 hex: 0x1p-126
pos_inf = inf hex: inf
NAN = nan hex: nan
yuanyue@1190200122-yuanyue: /mnt/hgfs/d$

```

(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

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     printf("1.0 / Float_MIN = %g\n", 1.0/0x1p-149);
6     printf("1.0 / 0 = %f\n", 1.0/0.0);
7     return 0;
8 }

```

```

yuanyue@1190200122-yuanyue: /mnt/hgfs/d$ gcc float0.c
yuanyue@1190200122-yuanyue: /mnt/hgfs/d$ ./a.out
1.0 / Float_MIN = 7.13624e+44
1.0 / 0 = inf
yuanyue@1190200122-yuanyue: /mnt/hgfs/d$

```

(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^{-149} ____；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^{104} ____。

最小正数变成十进制科学记数法，最可能精确到多少 ____ 1.401298×10^{-45} ____

最大正数变成十进制科学记数法，最可能精确到多少 ____
2.028241*10³¹_____

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。

浮点数储存时实际储存数字可能会有误差，因此不能直接比较大小，这个时候我们需要设置一个阈值 `eps`，如果两个数字的差值大于阈值，则可以直接比较，否则我们认为这两个数字相等。

第 8 章 舍尾平衡的讨论

8.1 描述可能出现的问题

当我们对浮点数进行应用记录时，通常会需要进行舍位操作，这个时候就会出现误差，而当这些误差积累的足够多时就会出现一个较大较明显的误差，使得我们得到的一些计算数值与实际的数值有较大误差，从而出现一些不可避免的错误。

8.2 给出完美的解决方案

我们可以通过使用更高精度的数据类型来进行计算，比如我们使用 `double` 或者 `long double`，其次我们也可以使用十进制小数的科学计数法来进行储存，用一个数组将所有的十进制数字储存下来，然后用一个有符号整型储存指数可以尽可能的提高精确度。

第 9 章 总结

9.1 请总结本次实验的收获

更加清楚了反汇编的过程以及应用，学会了使用 `gdb`, `objdump`, `readelf` 等工具，对于计算机的内部构造以及实现原理有了更多的了解，更了解了汇编语言的指令以及语法，了解更多浮点数的储存，对于机器指令以及更深层次的东西有了一个更好的认知，这在我们学习计算机系统上是十分重要的。

9.2 请给出对本次实验内容的建议

一些实验任务的要求叙述的不是很明确，总是看不懂需要干什么。

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>.