

# 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

学 号 1190200717

班 级 1903008

学 生 梁浩

指 导 教 师 吴锐

实 验 地 点 G709

实 验 日 期 2021/4/13

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（1 分）.....	- 7 -
3.2 数据的位置-地址（2 分）.....	- 7 -
3.3 MAIN 的参数分析（2 分）.....	- 9 -
3.4 指针与字符串的区别（2 分）.....	- 9 -
第 4 章 深入分析 UTF-8 编码.....	- 10 -
4.1 提交 UTF8LEN.C 子程序.....	- 10 -
4.2 C 语言的 STRCMP 函数分析.....	- 10 -
4.3 讨论：按照姓氏笔画排序的方法实现.....	- 10 -
第 5 章 数据变换与输入输出.....	- 11 -
5.1 提交 CS_ATOI.C.....	- 11 -
5.2 提交 CS_ATOF.C.....	- 11 -
5.3 提交 CS_ITOA.C.....	- 11 -
5.4 提交 CS_FTOA.C.....	- 11 -
5.5 讨论分析 OS 的函数对输入输出的数据有类型要求吗.....	- 11 -
第 6 章 整数表示与运算.....	- 12 -
6.1 提交 FIB_DG.C.....	- 12 -
6.2 提交 FIB_LOOP.C.....	- 12 -
6.3 FIB 溢出验证.....	- 12 -
6.4 除以 0 验证：.....	- 12 -
第 7 章 浮点数据的表示与运算.....	- 13 -
7.1 正数表示范围.....	- 13 -
7.2 浮点数的编码计算.....	- 13 -

7.3 特殊浮点数值编码.....	- 14 -
7.4 浮点数除 0.....	- 14 -
7.5 FLOAT 的微观与宏观世界.....	- 14 -
7.6 讨论：任意两个浮点数的大小比较.....	- 15 -
<b>第 8 章 舍位平衡的讨论.....</b>	<b>- 16 -</b>
8.1 描述可能出现的问题.....	- 16 -
8.2 给出完美的解决方案.....	- 16 -
<b>第 9 章 总结.....</b>	<b>- 17 -</b>
9.1 请总结本次实验的收获.....	- 17 -
9.2 请给出对本次实验内容的建议.....	- 17 -
<b>参考文献.....</b>	<b>- 18 -</b>

## 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

处理器: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40GHz  
已安装的内存(RAM): 8.00GB(7.81GB 可用)  
系统类型: 64 位操作系统, 基于 x64 的处理器

#### 1.2.2 软件环境

Windows 10 家庭中文版; VirtualBox 6.1; Ubuntu 20.04

#### 1.2.3 开发工具

Visual Studio 2019; vim+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  
Char /short int/int/long/float/double/long long/long double/指针
- 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

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）

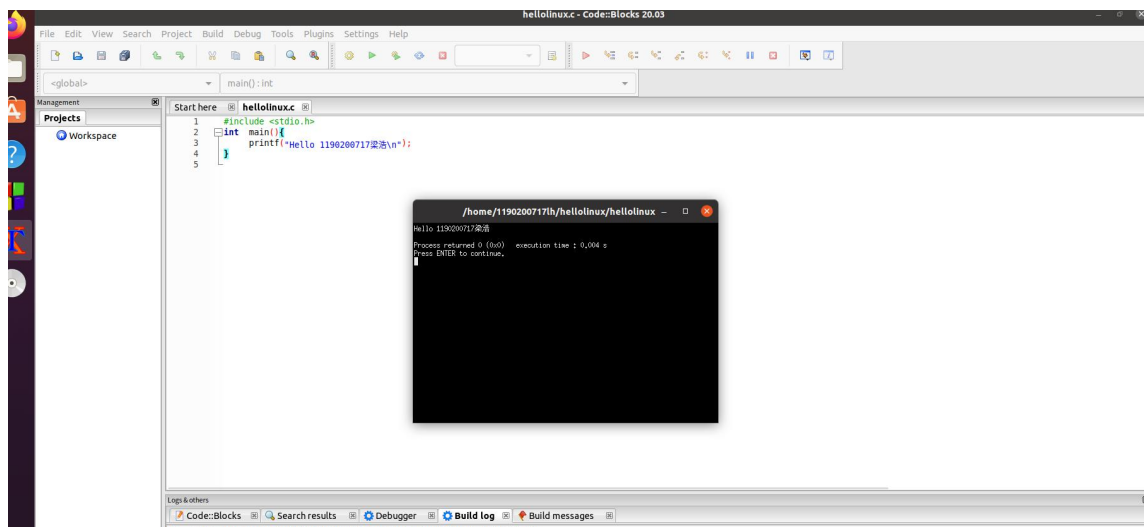
按步骤写出 `float` 数-1.1 在内存从低到高地址的字节值-16 进制

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

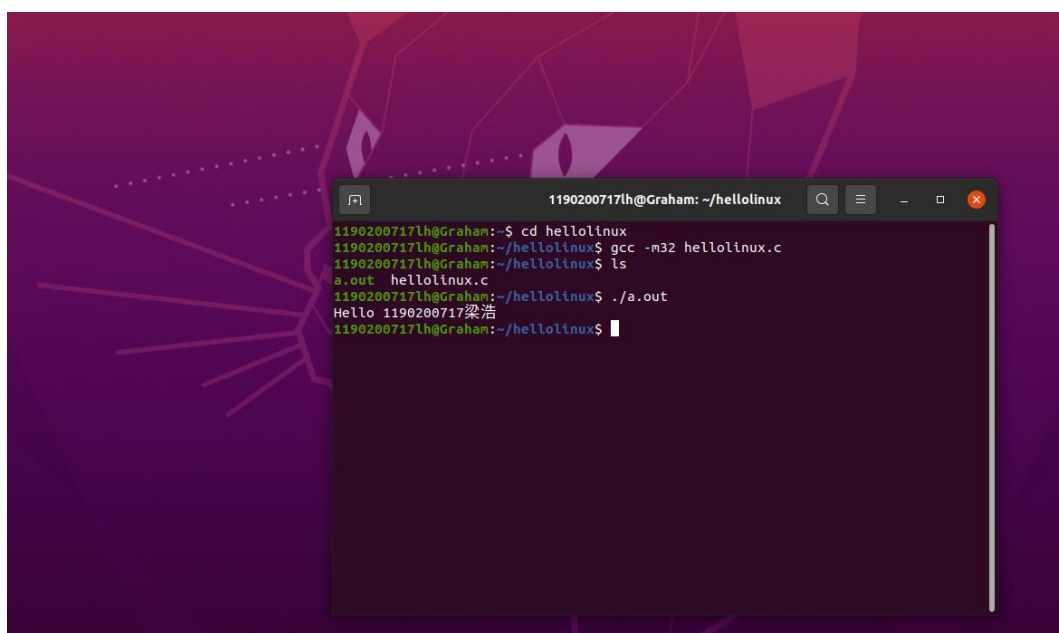
CodeBlocks 运行界面截图：编译、运行 hellolinux.c



### 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 (1 分)

	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	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

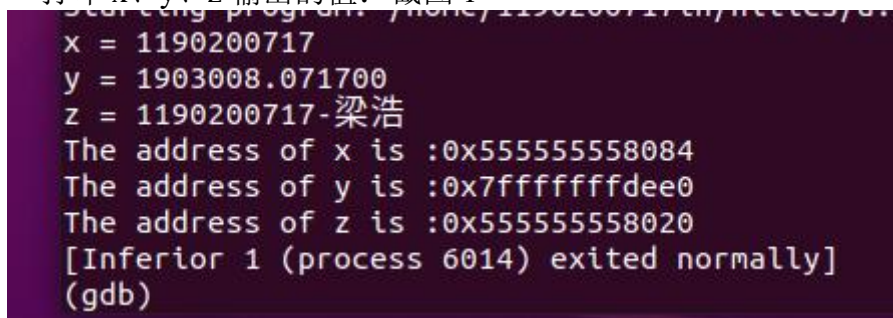
C 编译器对 sizeof 的实现方式:

sizeof 是 C 语言的一种单目操作符，并不是函数。sizeof 操作符以字节形式给出了其操作数的存储大小。操作数可以是一个表达式或括在括号内的类型名。操作数的存储大小由操作数的类型决定。sizeof 是获取了数据类型在内存中所占用的存储空间，以字节为单位来计数。

C/C++中，sizeof()只是运算符号，在编译的时候确定其大小。

### 3.2 数据的位置-地址 (2 分)

打印 x、y、z 输出的值：截图 1



```

Starting program: /home/1190200717/notes/01/...
x = 1190200717
y = 1903008.071700
z = 1190200717-梁浩
The address of x is :0x555555558084
The address of y is :0x7fffffffdee0
The address of z is :0x555555558020
[Inferior 1 (process 6014) exited normally]
(gdb)

```

反汇编查看 x、y、z 的地址，每字节的内容：截图 2，标注说明

```
The address of x is :0x55555558084
The address of y is :0x7fffffffdee0
The address of z is :0x55555558020
```

(x、y、z 的地址)

```
(gdb) x /4xb 0x55555558084
0x55555558084 <x>:      0x8d      0x05      0xf1      0x46
```

(x 每字节的内容)

```
(gdb) x /8xb 0x7fffffffdee0
0x7fffffffdee0: 0x63      0xee      0x5a      0x12      0xa0      0x09      0x3d      0x41
```

(y 每字节的内容)

```
(gdb) x /20xb 0x55555558020
0x55555558020 <z.2514>:      0x31      0x31      0x39      0x30      0x32      0x30      0x30      0x37
0x55555558028 <z.2514+8>:      0x31      0x37      0x2d      0xe6      0xa2      0x81      0xe6      0xb5
0x55555558030 <z.2514+16>:      0xa9      0x00      0x00      0x00      0x00      0x00      0x00      0x00
```

(z 每字节的内容)

反汇编查看 x、y、z 在代码段的表示形式。截图 3，标注说明

```
X:
0000000000004020 <x>:
4020:      8d 05 f1 46 00 00      lea     0x46f1(%rip),%eax      # 8717 <__TMC_END__+0x466f>
```

```
y:
2070:      63 ee      movslq  %esi,%ebp
2072:      5a      pop     %rdx
2073:      12      .byte  0x12
2074:      a0      .byte  0xa0
2075:      09      .byte  0x9
2076:      3d      .byte  0x3d
2077:      41      rex.B
```

```
Z:
0000000000004040 <z.2319>:
4040:      31 31      xor     %esi,(%rcx)
4042:      39 30      cmp     %esi,(%rax)
4044:      32 30      xor     (%rax),%dh
4046:      30 37      xor     %dh,(%rdi)
4048:      31 37      xor     %esi,(%rdi)
404a:      2d e6 a2 81 e6      sub     $0xe681a2e6,%eax
404f:      b5 a9      mov     $0xa9,%ch
...
```

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：对于 C/C++，数值型常量的存储有两种情况，具体要看是否被 `const` 修饰，如果被 `const` 修饰，数值型常量就存储在内存的常量区，如果没有，编译器会通过立即数来实现，并不会存储下来；变量是存储在栈区，由编译器自动分配释放。

字符串常量与变量在存储空间上的区别是：字符串常量存储在常量区，变量存



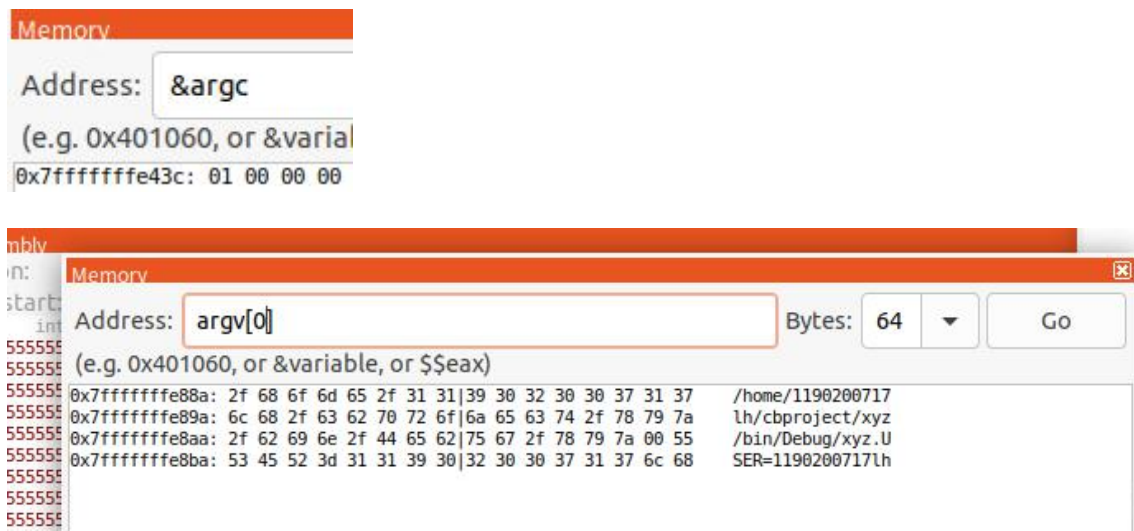
储在栈区，由编译器自动分配释放。

常量表达式在计算机中处理方法是：常量表达式将在编译时而不是运行时计算，值不能修改。

### 3.3 main 的参数分析 (2 分)

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

```
The address of x is :0x555555558084
The address of y is :0x7fffffffdee0
The address of z is :0x555555558020
```



### 3.4 指针与字符串的区别 (2 分)

cstr 的地址与内容截图，pstr 的内容与截图，截图 5

cstr 的地址与内容截图：

```
(gdb) x /20xb 0x555555558020
0x555555558020 <cstr>: 0x31 0x31 0x39 0x30 0x32 0x30 0x30 0x37
0x555555558028 <cstr+8>: 0x31 0x37 0x2d 0xe6 0xa2 0x81 0xe6 0xb5
0x555555558030 <cstr+16>: 0xa9 0x00 0x00 0x00
The address of pstr is :0x7fff389520e0
pstr的内容是 : 1190200717-梁浩
```

尝试着修改 pstr 后：

```
printf("The address of cstr is :%p\n",cstr);
printf("The address of pstr is :%p\n",&pstr);
printf("pstr的内容是: %s\n",pstr);
strcpy(cstr,"1903008.0717");
printf("修改后cstr的内容是: %s\n",cstr);
strcpy(pstr,"1903008.0717");

printf("修改后pstr的内容是: %s\n",pstr);
```

The screenshot shows the output of the program. It displays the addresses of `cstr` and `pstr`, and the content of `pstr`. The output shows that `pstr` contains the string `1190200717-梁浩`. The program then attempts to modify `pstr` using `strcpy`, but it results in a segmentation fault (core dumped).

pstr 修改内容会出现什么问题：无法修改，报错(Segmentation fault)

## 第 4 章 深入分析 UTF-8 编码

### 4.1 提交 utf8len.c 子程序

见附件

### 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 按照汉字对应的 unicode 编码大小对汉字进行排，首先比较姓的 unicode 编码大小，若一样，则继续比较下一位的 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？

可以考虑利用具有前缀性的编码来对各个笔画进行编码、排序。在对姓氏进行比较时，只需要比较它们笔画对应的编码大小即可。

## 第 5 章 数据变换与输入输出

### 5.1 提交 `cs_atoi.c`

见附件

### 5.2 提交 `cs_atof.c`

见附件

### 5.3 提交 `cs_itoa.c`

见附件

### 5.4 提交 `cs_ftoa.c`

见附件

### 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：

有要求。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，分别为：

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

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

`read` 函数中有一个 `size_t` 的输入参数，返回值是 `ssize_t` 类型。在 64 位系统中，`size_t` 被定义为 `unsigned long`，而 `ssize_t` 被定义为 `signed size_t`。`read` 函数返回类型是有符号是因为出错时需要返回-1。

## 第 6 章 整数表示与运算

### 6.1 提交 fib\_dg.c

见附件

### 6.2 提交 fib\_loop.c

见附件

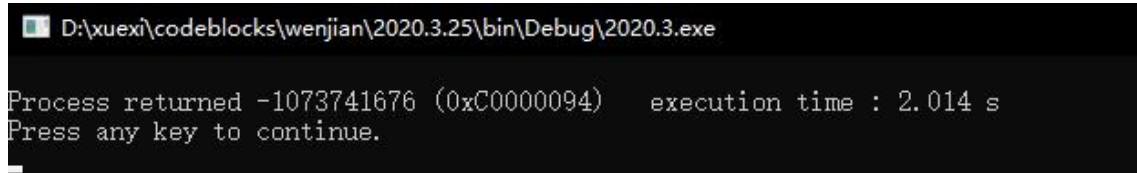
### 6.3 fib 溢出验证

int 时从 n= 47 时溢出，long 时 n= 47(long 长度为 4 字节) / 93(long 长度为 8 字节) 时溢出。

unsigned int 时从 n= 48 时溢出，unsigned long 时 n= 94 时溢出。

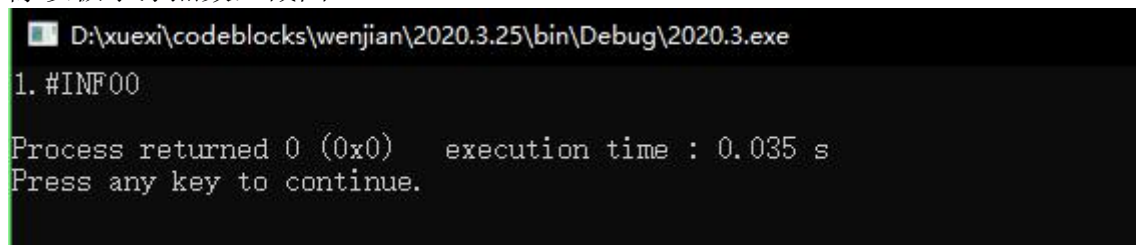
### 6.4 除以 0 验证：

除以 0：截图



```
D:\xuexi\codeblocks\wenjian\2020.3.25\bin\Debug\2020.3.exe
Process returned -1073741676 (0xC0000094)   execution time : 2.014 s
Press any key to continue.
```

除以极小浮点数，截图：



```
D:\xuexi\codeblocks\wenjian\2020.3.25\bin\Debug\2020.3.exe
1. #INF00
Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

## 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）

对于 float 类型，

1) 当  $s = 0$ ,  $\text{exp} = 1111\ 1110$ ,  $\text{frac} = 11111\ 11111\ 11111\ 11111\ 111$  时取最大正数，最大数为  $(2 - 2^{-23}) * 2^{127}$

2) 当  $s = 0$ ,  $\text{exp} = 0000\ 0000$ ,  $\text{frac} = 00000\ 00000\ 00000\ 00000\ 001$  时取最小正数，最小值为  $2^{-149}$

对于规格化数，最小的正数是  $s = 0$ ,  $\text{exp} = 0000\ 0001$ ,  $\text{frac} = 00000\ 00000\ 00000\ 00000\ 000$ , 最小值为  $2^{-126}$

对于 double 类型(符号位 1 位，阶码 11 位，尾数 52 位)

1) 当  $s = 0$ ,  $\text{exp} = 111\ 1111\ 1110$ ,  $\text{frac} = 11111 \sim 1111$  (52 个 1) 时，取最大正数，最大整数为:  $(2 - 2^{-52}) \cdot 2^{1023}$

2) 当  $s = 0$ ,  $\text{exp} = 000\ 0000\ 0000$ ,  $\text{frac} = 00000 \sim 0001$  (51 个 0) 时，取最小正数，最小值为:  $2^{-1074}$

对于规格化数，最小的正数为  $s = 0$ ,  $\text{exp} = 000\ 0000\ 0001$ ,  $\text{frac} = 00000 \sim 0000$  (52 个 0), 最小值为  $2^{-1022}$

### 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

$$-1.1 = -1.00011001100110011001101B \times 2^0$$

$$s = 1$$

$$\text{frac} = 00011001100110011001101$$

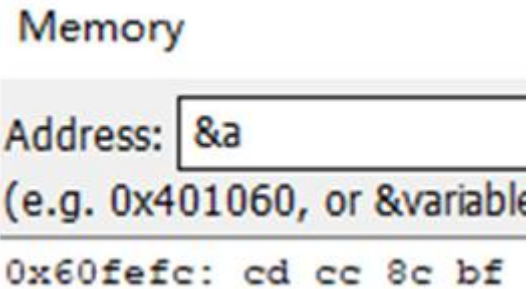
$$E = 0$$

$$\text{exp} = 0 + 127 = 127 = 01111111B$$

所以，-1.1 的机器数表示为 1011 1111 1000 1100 1100 1100 1100 1101，转化为 16 进制表示为 BF8CCCCD，

对应的在内存中从低地址字节到高地址字节的 16 进制数值为 CD CC 8C BF

(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。



### 7.3 特殊浮点数值的编码

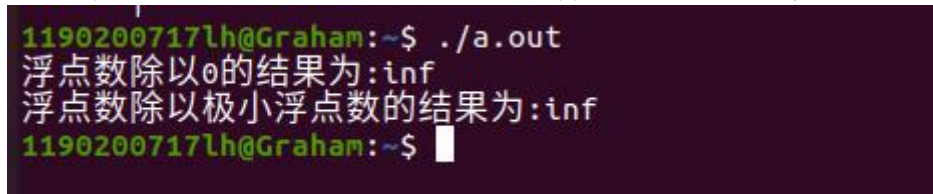
(1) 构造多 float 变量，分别存储+0-0，最小浮点正数，最大浮点正数、最小正的规格化浮点数、正无穷大、Nan,并打印最可能的精确结果输出（十进制/16 进制）。截图。



(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



(2) 提交子程序 float0.c

### 7.5 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):

规格化数区域 (按正半轴计算):

最小: 0 0000 0001 00000 00000 00000 00000 000

十进制表示为:  $2^{-126}$

最大: 0 1111 1110 11111 11111 11111 11111 111

十进制表示为:  $(2-2^{-23}) \cdot 2^{127}$

表示的浮点个数:  $254 \cdot 2^{23}$

密度为:  $\frac{(2-2^{-23}) \cdot 2^{127} - 2^{-126}}{254 \cdot 2^{23}} = 1.5970 \cdot 10^{-29}$

非规格化数区域 (按正半轴计算):

最小: 0 0000 0000 00000 00000 00000 00000 000

十进制表示为: 0

最大: 0 0000 0000 11111 11111 11111 11111 111

十进制表示为:  $(1-2^{-23}) \cdot 2^{-126}$

表示的浮点个数:  $2^{23}$

密度为:  $(1-2^{-23}) \cdot 2^{-149} = 1.401298297 \cdot 10^{-45}$

最小正数变成十进制科学记数法, 最可能精确到多少:  $2^{-149} = 1.401298 \cdot 10^{-45}$

最大正数变成十进制科学记数法, 最可能精确到多少:  $3.402823 \cdot 10^{38}$

## 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:

判断要比较的两个浮点数之差的绝对值是否小于一个很小的浮点数, 这里设很小的浮点数为 EPS, 若绝对值小于 EPS, 则认为两个浮点数相等, 否则不相等

原因:

计算机中浮点数的表示是有误差的, 所以在比较它们大小的时候应该比较它们的差值是否接近 0, 如果在一定范围内接近 0 就可以认为它们相等。

## 第 8 章 舍位平衡的讨论

### 8.1 描述可能出现的问题

舍位平衡主要采取四舍五入的方法，这就会产生误差，误差会随着计算次数的增加而逐渐累积，举个例子：

$0.5+0.5=1$ ，采用四舍五入的方法后会变成  $1+1=2$ ，和实际值相差 1

### 8.2 给出完美的解决方案

主要思想：

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

具体实现：

平衡差=求和后舍位值 - 舍位后求和值

最小调整值：舍位后最小精度的单位值

若舍位后总和变小，则需要将数据调大，最小调整值是 1

若舍位后总和变大，则需要将数据调小，最小调整值是-1

舍位平衡的调整只针对绝对值比较大的数据，这样使相对偏差比较小。

用平衡差的绝对值除以最小调整值得到需要调整数据的个数  $n$ ，然后对原始数据的绝对值进行排序，从大到小的  $n$  个数加上最小调整值，最后将调整后的值和未调整的值重新求和。

举例说明：

对于 3.4, 1.3, 8.9, 7.1，取整到个位，求和后舍位值 = 21，舍位后求和值 = 20  
平衡差 = 1, 用平衡差的绝对值 1 除以最小调整值 1 得到需要调整数据的个数  $n=1$ ，  
对四个数舍位后绝对值从大到小进行排序，为 9, 7, 3, 1，对第一个数加上最小调整值 1，变为 10, 7, 3, 1，相加后的值为 21，实现了舍位平衡。



## 第 9 章 总结

### 9.1 请总结本次实验的收获

本次实验我学会了如何在 Ubuntu 利用 codeblocks 编译和运行代码，如何利用 gcc 和 objdump 来查看反汇编，获取不同类型的变量在内存中的存储位置。通过实际操作，我还知道了指针和字符串的区别。同时我还了解了 UTF-8 编码，学会用 C 语言实现字符、浮点数、整数之间的相互转换。对浮点数的讨论加深了我对 IEEE-754 规定下浮点数表示的理解。

### 9.2 请给出对本次实验内容的建议

PPT 上的实验要求最好可以再明确一些

注：本章为酌情加分项。

## 参考文献

- [1] 深入理解计算机系统