

哈尔滨工业大学

实验报告

实验（一）

题 目 系统漫游与数据表示

专 业 计算机科学与技术

学 号 2022113573

班 级 2203101

学 生 张宇杰

指导教师 史先俊

实验地点 管理 712

实验日期 2023/10/19

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	- 5 -
2.1 UBUNTU 下 CODEBLOCKS 安装	- 5 -
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	- 7 -
3.3 MAIN 的参数分析	- 12 -
3.4 指针与字符串的区别	- 13 -
第 4 章 深入分析 UTF-8 编码	- 15 -
4.1 提交 UTF8LEN.C 子程序	- 15 -
4.2 C 语言的 STRCMP 函数分析	- 15 -
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_ATOL.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	- 19 -
第 7 章 浮点数据的表示与运算	- 21 -

7.1 手动 FLOAT 编码:	- 21 -
7.2 特殊 FLOAT 数据的处理	- 21 -
7.3 验证浮点运算的溢出	- 22 -
7.4 类型转换的坑	- 22 -
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	- 23 -
7.7 讨论 3: FLOAT 能精确表示几个 1 元内的钱呢	- 24 -
7.8 FLOAT 的微观与宏观世界	- 24 -
7.9 讨论: 浮点数的比较方法	- 24 -
第 8 章 程序运行分析	- 25 -
8.1 隐式类型转换	- 25 -
8.2 浮点数的精度	- 26 -
第 9 章 舍尾平衡的讨论	- 28 -
9.1 描述可能出现的问题	- 28 -
9.2 给出完美的解决方案	- 28 -
第 10 章 总结	- 29 -
10.1 请总结本次实验的收获	- 29 -
10.2 请给出对本次实验内容的建议	- 29 -
参考文献	- 30 -

第 1 章 实验基本信息

1.1 实验目的

理解计算机软硬件系统的构成

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

1.2 实验环境与工具

1.2.1 硬件环境

12th Gen Intel(R) Core(TM) i5-12500H 3.10GHz + RAM 16.0GB

1.2.2 软件环境

Windows 11 64 位+ VMware® Workstation 16 Pro (16.2.4 build-20089737)

1.2.3 开发工具

Microsoft Visual Studio Community 2022 (64 位) - Current 版本 17.7.4

CodeBlocks 20.03, Vim + 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 hello.c（输出 Hello 学号-姓名！）

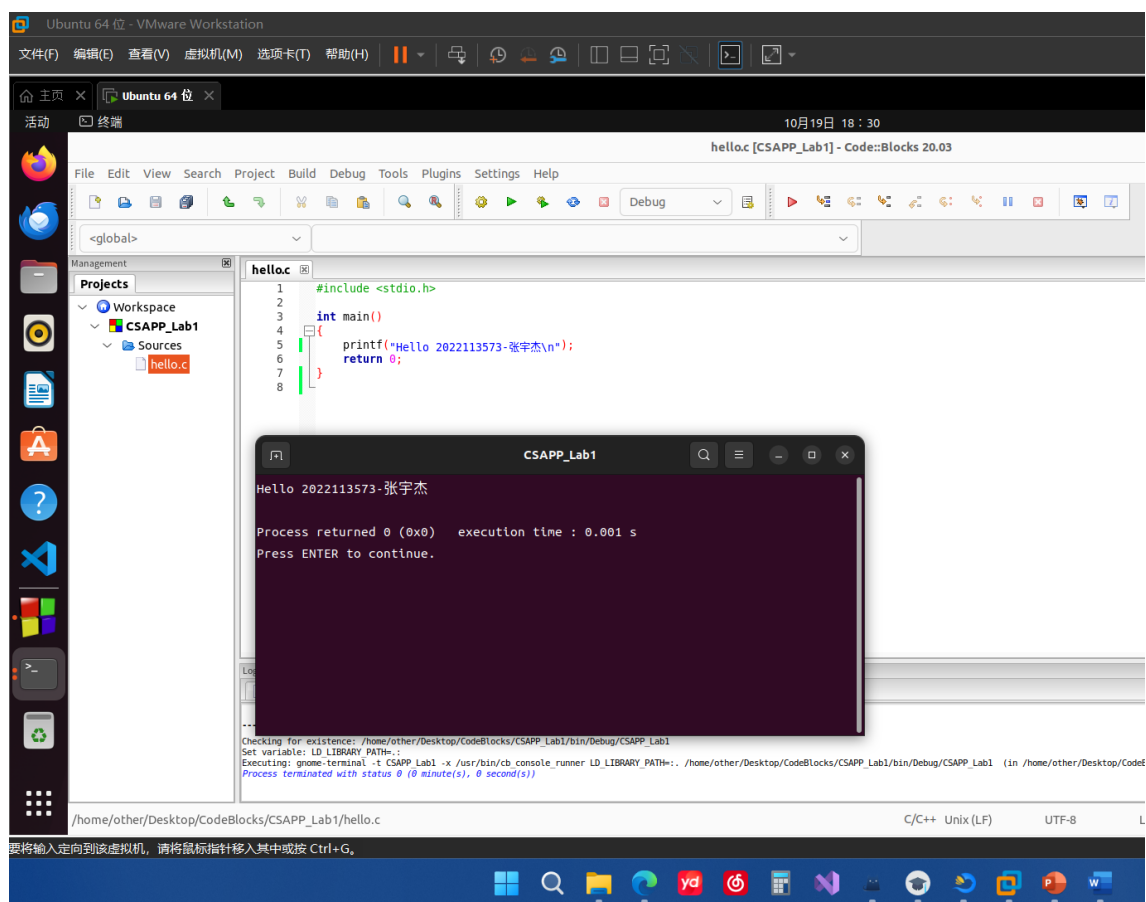


图 2-1 Ubuntu 下 CodeBlocks 截图

2.2 64 位 Ubuntu 下 32 位运行环境建立

在 Terminal 终端窗口下使用 gcc 的 32 位模式编译生成 hello。执行此文件。
Linux 及终端的截图。

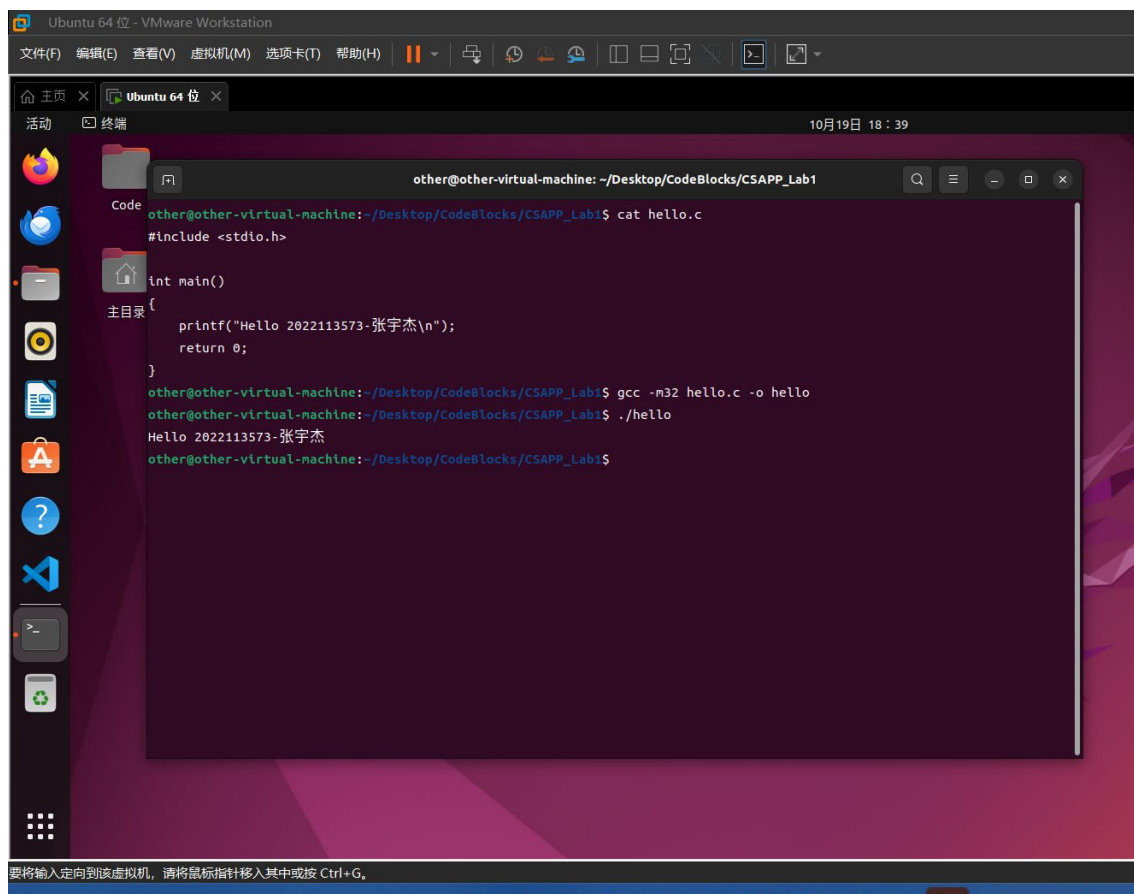


图 2-2 32 位模式运行成功证明！

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 在编译阶段就已经求值了，具体值跟编译器有关

3.2 数据的位置-地址

打印 x、y、z 输出的值：截图 1

```

1  #include <stdio.h>
2
3  int x = -2022113573;
4
5  int main()
6  {
7      float y = 350783200408150210;
8      static const char* z = "2022113573-张宇杰";
9
10     printf("%d\n", x);
11     printf("%f\n", y);
12     printf("%s\n", z);
13
14     return 0;
15 }

```

Microsoft Visual Studio 调试

```

-2022113573
350783200408150210.000000
2022113573-张宇杰

```

D:\VS2022\CSAPP_Lab1_1\x64\Debug\CSAPP_Lab1_1.exe (进程 25512) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

反汇编查看 x、y、z 的地址，每字节的内容：截图 2，标注说明

X 的地址

内存 1
地址: 0x00007FF72471C000 列: 自动

0x00007FF72471C000	db fe 78 87 00 00 00 00 b0 9b 71 24 f7 7f 00 00 00 00 00	?? x?... ?? q\$?.....
0x00007FF72471C013	00 00 00 00 00 00 00 00 00 00 00 00 54 ec 8a 17 82 8fT??..??
0x00007FF72471C026	ff ff ab 13 75 e8 7d 70 00 00 00 00 00 00 75 98 00 01	.. ?u?}p.....u? ...
0x00007FF72471C039	00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 01 00 00
0x00007FF72471C04C	00 00 00 00 ff ff ff ff 01 00 00 00 05 00 00 00 2e 00 00

程序员

8778 FEDB

HEX	8778 FEDB
DEC	-2,022,113,573
OCT	20 736 177 333
BIN	1000 0111 0111 1000 1111 1110 1101 1011

Y 的地址

内存 1
地址: 0x0000004274B6F614 列: 自动

0x0000004274B6F614	6f c7 9b 5c 32 00 00 00 00 00 00 00 00 00 9f 01 00	o??\2.....?..
0x0000004274B6F627	00 c8 03 ce b1 fc 7f 00 00 50 f6 b6 74 42 00 00 00 f9 2d	..?..?? ... P??tB ... ?-

Decimal	350783200408150210
---------	--------------------

32 bit float

Decimal (exact)	350783207563591680
Binary	0 10111001 001101111100011101101111
Hexadecimal	5C9BC76F

Z 指向内容的地址（即 z 本身）

采用 GB2312 编码

内存 1
地址: 0x00007FF7247198B0 列: 自动

0x00007FF7247198B0	32 30 32 32 31 31 33 35 37 33 2d d5 c5 d3 ee bd dc 00 00	2022113573-?????..
0x00007FF7247198C3	00 00 00 00 00 00 00 00 00 00 00 00 5f 41 72 67 4c 69_ArgLi
0x00007FF7247198D6	73 74 00 00 00 00 00 00 00 00 48 00 00 00 08 00 00 d0	st.....H.....?
0x00007FF7247198E9	9b 71 24 f7 7f 00 00 00 00 00 00 00 00 00 00 00 00 00	?q\$?.....

2022113573-张宇杰

字符编码: GB2312

☒ 十六进制带 \x 前缀☐ 十六进制大写

编码 ↓

\x32\x30\x32\x32\x31\x31\x33\x35\x37\x33\x2d\x2d\x5c\x5d\x3e\x2d\x2d

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: 00007FF72471C000h

Y: rbp + 4

Z: 00007FF72471C008h

```

printf("%d\n", x);
00007FF7247118A8 8B 15 52 A7 00 00 mov     edx,dword ptr [00007FF72471C000h]
00007FF7247118AE 48 8D 0D 8F 83 00 00 lea     rcx,[00007FF724719C44h]
00007FF7247118B5 E8 DB F8 FF FF call    00007FF724711195
printf("%f\n", y);
00007FF7247118BA F3 0F 5A 45 04 cvtss2sd xmm0,dword ptr [rbp+4]
00007FF7247118BF 0F 28 C8 movaps  xmm1,xmm0
00007FF7247118C2 66 48 0F 7E CA movq    rdx,xmm1
00007FF7247118C7 48 8D 0D 7A 83 00 00 lea     rcx,[00007FF724719C48h]
00007FF7247118CE E8 C2 F8 FF FF call    00007FF724711195
printf("%s\n", z);
00007FF7247118D3 48 8B 15 2E A7 00 00 mov     rdx,qword ptr [00007FF72471C008h]
00007FF7247118DA 48 8D 0D 6B 83 00 00 lea     rcx,[00007FF724719C4Ch]
00007FF7247118E1 E8 AF F8 FF FF call    00007FF724711195

```

x 与 y 在 汇编 阶段转换成补码与 ieee754 编码。

理由:

对 x:

xyz_test.i (预处理)

```

# 3 "xyz_test.c"
int x = -2022113573;

int main()

```

xyz_test.s (编译生成的汇编代码)

```

xyz_test.s
~/Desktop/Code/C_single/CSAPP_Lab1
5      .align 4
6      .type x, @object
7      .size x, 4
8 x:
9      .long -2022113573
10     .section .rodata
11 .LC1:

```

xyz_test.o.txt (对汇编生成的机器码 xyz_test.o 用 objdump 命令进行反编译的输出)

```

xyz_test.o.txt
~/Desktop/Code/C_single/CSAPP_Lab1 保存(S)
xyz_test.txt x xyz_test.o.txt
13 Contents of section .data:
14 0000 dbfe7887 ..X.
15 Contents of section .rodata:
16 0000 25640a00 25660a00 32303232 31313335 %d..%f..20221135
17 0010 37332de5 bca0e5ae 87e69db0 00000000 73-.....
18 0020 6fc79b5c o..
程序员
8778 FEDB
HEX 8778 FEDB
DEC -2,022,113,573
OCT 20 736 177 333
BIN 1000 0111 0111 1000 1111 1110 1101 1011

```

xyz_test.txt (对链接生成的可执行文件 xyz_test 用 objdump 命令进行反编译的输出)

```

xyz_test.txt
~/Desktop/Code/C_single/CSAPP_Lab1 保存(S)
142 Contents of section .data:
143 4000 00000000 04400000 dbfe7887 10200000 .....@....X...
144 Contents of section .comment:
145 0000 4743433a 20285562 756e7475 2031312e GCC: (Ubuntu 11.

```

对 y:

xyz_test.i (预处理)

```

# 3 "xyz_test.c"
int x = -2022113573;

int main()
{
    float y = 350783200408150210;
}

```

xyz_test.s (编译生成的汇编代码)

```

xyz_test.s
~/Desktop/Code/C_single
63     .section .rodata
64     .align 4
65 .LC0:
66     .long 1553713007
67     .ident "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"

```

三	程序员
1,553,713,007	
HEX	5C9B C76F
DEC	1,553,713,007
OCT	13 446 743 557
BIN	0101 1100 1001 1011 1100 0111 0110 1111

而 5C9BC76F 正好是 350783200408150210 转换为浮点数后对应的 ieee754 编码

Decimal	350783200408150210
32 bit float	
Decimal (exact)	350783207563591680
Binary	0 10111001 00110111100011101101111
Hexadecimal	5C9BC76F

xyz_test.o.txt (对汇编生成的机器码 xyz_test.o 用 objdump 命令进行反编译的输出)

```

14 0000 d0fe7887                ..X.
15 Contents of section .rodata:
16 0000 25640a00 25660a00 32303232 31313335 %d..%f..20221135
17 0010 37332de5 bca0e5ae 87e69db0 00000000 73-.....
18 0020 6fc79b5c                o..\
19 Contents of section .data.rel.local:

```

xyz_test.txt (对链接生成的可执行文件 xyz_test 用 objdump 命令进行反编译的输出)

```

98 Contents of section .rodata:
99 2000 03000000 01000200 25640a00 25660a00 .....%d..%f..
100 2010 32303232 31313335 37332de5 bca0e5ae 2022113573-.....
101 2020 87e69db0 00000000 6fc79b5c                .....o..\
102 Contents of section .eh_frame_hdr:

```

数值型常量与变量在存储空间上的区别是：

给局部变量初始化的整数常量，作为立即数，直接存储在指令中；给其他变量初始化的整数常量，存储在.data 节

实型常量存储在.rodata 节

而数值型局部变量存储在栈区，全局变量与静态变量存储在静态区

字符串常量与变量在存储空间上的区别是：

字符串常量在编译时被确定，存储在.rodata 节中

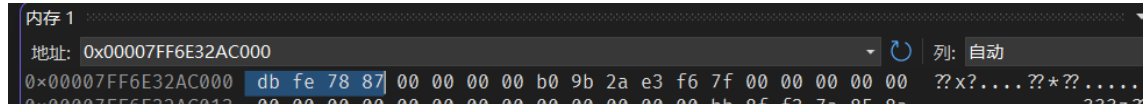
而字符串变量是字符数组或指向字符的指针。字符串局部变量在运行时创建，存储在栈区

常量表达式在计算机中处理方法是：在编译时对其进行求值，并存储在相应的区中(.rodata, 立即数等)

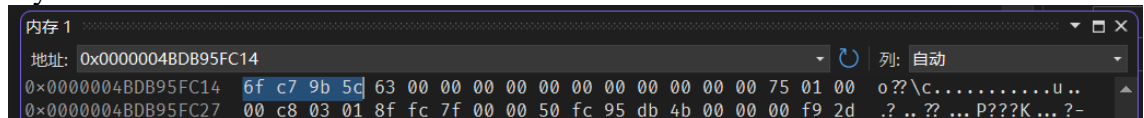
3.3 main 的参数分析

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

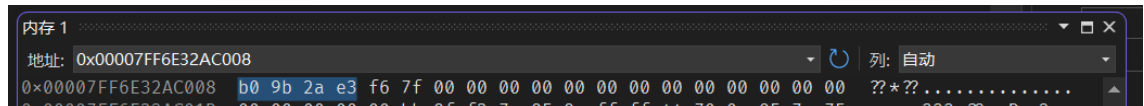
&x



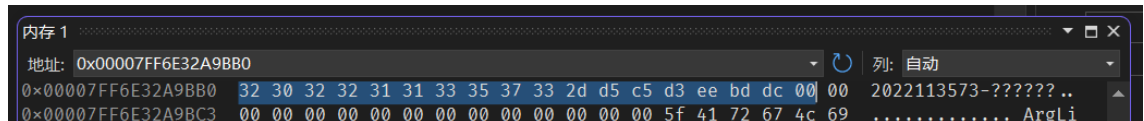
&y



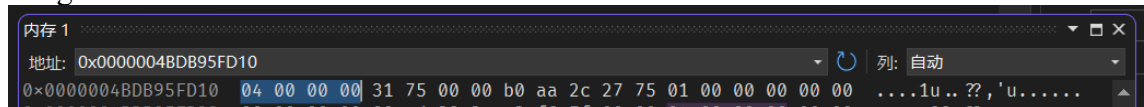
&z



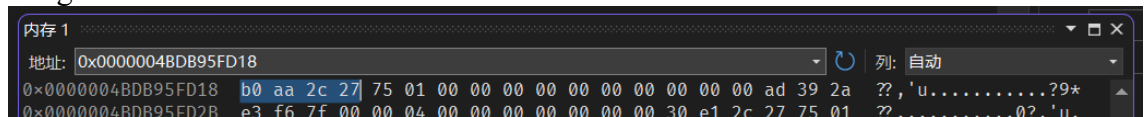
z



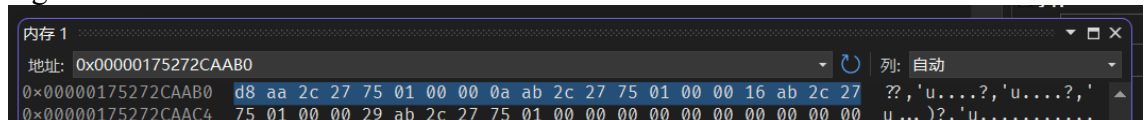
&argc



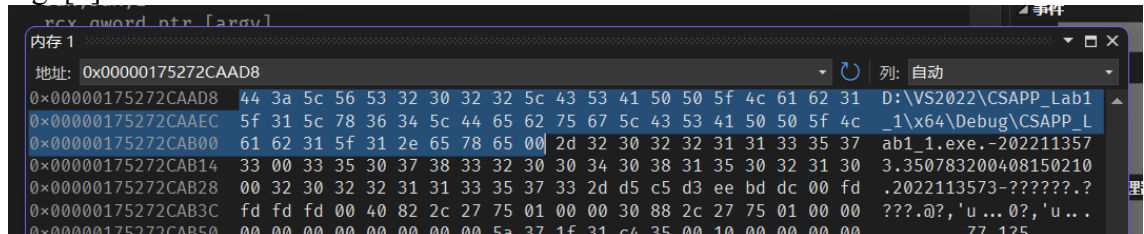
&argv



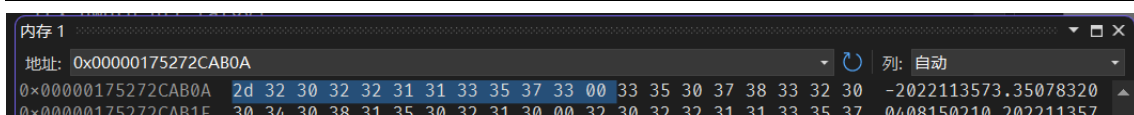
argv



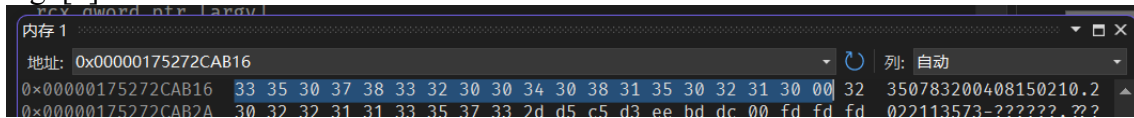
argv[0]



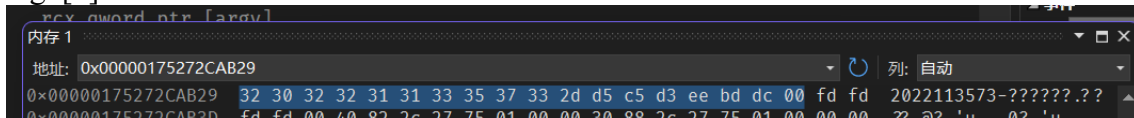
argv[1]



argv[2]



argv[3]



3.4 指针与字符串的区别

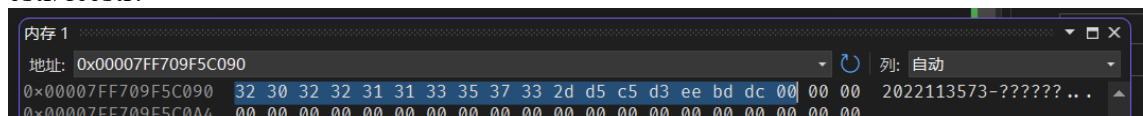
```

1  #include <stdio.h>
2  #include <string.h>
3  #pragma warning(disable: 4996)
4
5  char cstr[100] = "2022113573-张宇杰";
6  char* pstr = "2022113573-张宇杰";
7
8  int main()
9  {
10     printf("cstr: %s\n", cstr);
11     printf("pstr: %s\n", pstr);
12     strcpy(cstr, "3507832004081021X");
13     strcpy(pstr, "3507832004081021X");
14
15     return 0;
16 }

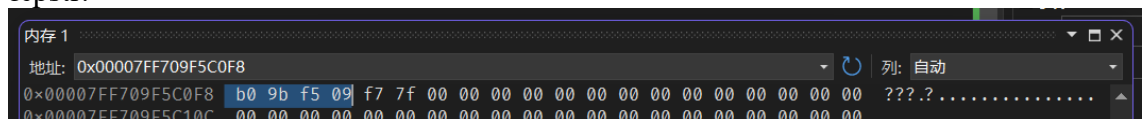
```

cstr 的地址与内容截图，pstr 的内容与截图，截图 5

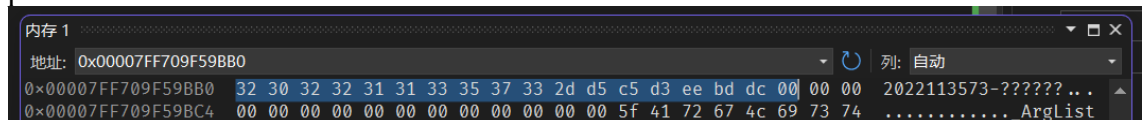
cstr/&cstr:



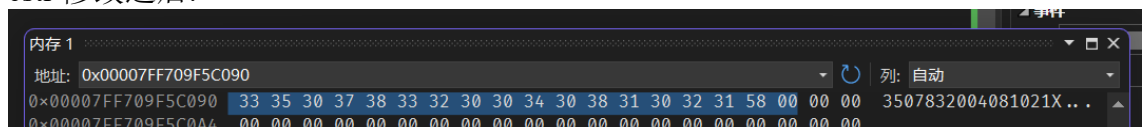
&pstr:



pstr:



cstr 修改过后:



pstr 修改后:



pstr 修改内容会出现什么问题: pstr 指向了一个位于.rodata 区(常量区)的常量字符串, 是只读的, 修改会引发段错误

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

通过查看 `strcmp` 的源码, 我们知道, `strcmp` 只是单纯的将两个字符数组进行逐字节比较。

故我们知道, 两个汉字字符串进行比较, 仅与他们的编码有关, 谁编码的值大谁就更大。

如 GB2312 编码, 其中的一级汉字是按照拼音排序的, 所以对一级汉字使用 `strcmp` 比较就是在比较他们拼音顺序的先后。而对于二级汉字, 编码按照部首/笔画进行排序。所以, 将一、二级汉字之间或二、二级汉字之间使用 `strcmp` 比较是没有实际意义的。

再如 UTF-8 编码, 虽说 UTF-8 中汉字的先后顺序参照了康熙字典, 但实际上汉字在其中的排序以及分布都比较乱, 使用 `strcmp` 进行比较常常不能达到预期的结果。不过我们可以认为, 对于 UTF-8 编码, `strcmp` 大体上是按照康熙字典来排序的。

4.3 讨论: 按照姓氏笔画排序的方法实现

可以在每个汉字的编码都额外加一个字节, 记录笔画数(0 至 255), 在进行汉字排序的时候就按照其编码中存储的笔画数来排序。

可以在设计编码的时候就将汉字以笔画顺序排好, 笔画少的汉字对应编码的码位就越小。对于相同笔画的汉字, 可以按照一定的顺序进行排序, 如偏旁、拼音等。这样在排序时就能自动按照笔画顺序排好。

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

我认为没有。

论述如下：

请看 `read/write` 的函数原型：

```
int __cdecl read(int _FileHandle, void* _DstBuf, unsigned int _MaxCharCount);  
int __cdecl write(int _FileHandle, void const* _Buf, unsigned int _MaxCharCount);
```

它们的 `buffer` 参数为指向 `void` 类型的指针，而通过 `void` 类型的指针去访问数据，会失去原数据的类型信息，这意味着它们可以接受任何类型的数据。

第 6 章 整数表示与运算

6.1 提交 fib_dg.c

6.2 提交 fib_loop.c

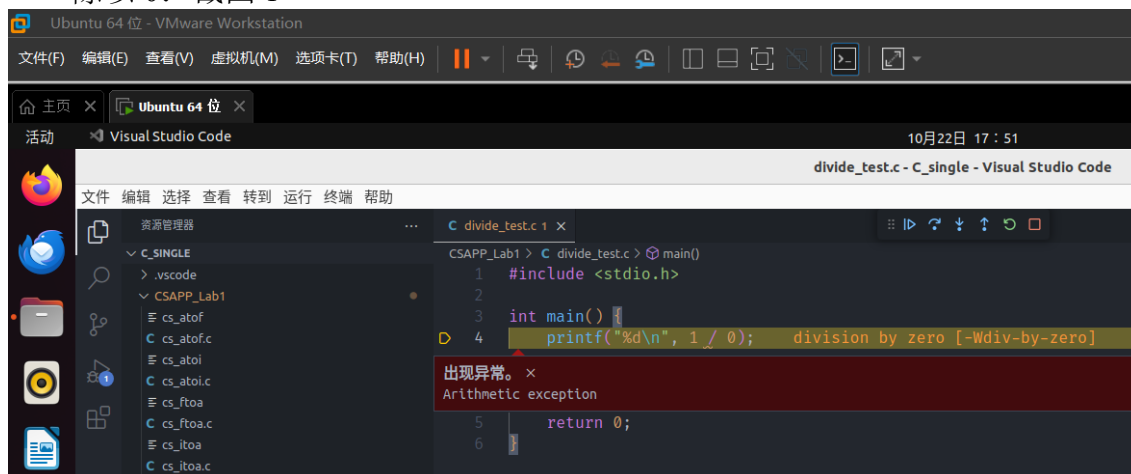
6.3 fib 溢出验证

int 时从 n= 47 时溢出, long 时 n= 93 时溢出。

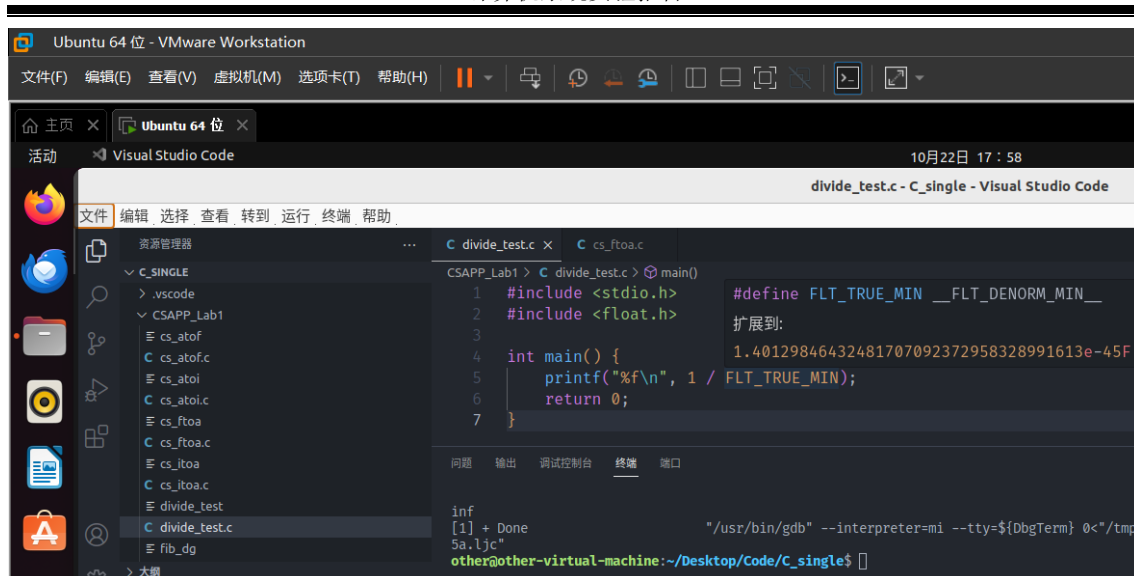
unsigned int 时从 n= 48 时溢出, unsigned long 时 n= 94 时溢出。

6.4 除以 0 验证:

除以 0: 截图 1



除以极小浮点数, 截图:



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：

```
C:\Users\14276>date  
当前日期: 2023/10/22 周日  
输入新日期: (年月日) 9999/12/31  
系统无法接受输入的日期。  
输入新日期: (年月日) |
```

```
other@other-virtual-machine:~$ date  
2023年 10月 22日 星期日 21:25:21 CST  
other@other-virtual-machine:~$ date -s "9999-12-31 23:59:59"  
date: 无法设置日期: 无效的参数  
9999年 12月 31日 星期五 23:59:59 CST  
other@other-virtual-machine:~$
```

```
#include<stdio.h>
#include<time.h>
#pragma warning(disable : 4996)

int main() {
    time_t seconds = time(NULL);
    struct tm* tLocal = gmtime(&seconds);

    printf("old year: %d\n", tLocal->tm_year);
    tLocal->tm_year = 9999 - 1900;
    printf("new year: %d\n", tLocal->tm_year);

    time_t t = mktime(tLocal);
    printf("ctime: %s\n", ctime(&t));
    return(0);
}
```

Microsoft Visual Studio 调试

old year: 123
new year: 8099
ctime: (null)

D:\VS2022\CSAPP_Lab1_1\D
按任意键关闭此窗口...

```
C test.c x
CSAPP_Lab1 > C test.c > main()
1  #include<stdio.h>
2  #include<time.h>
3
4  int main() {
5      time_t ttt = 253402271999;
6      printf("ctime: %s\n", ctime(&ttt));
7      ttt = 253402272000;
8      printf("ctime: %s\n", ctime(&ttt));
9      return(0);
10 }
```

问题 输出 调试控制台 终端 端口

ctime: Fri Dec 31 23:59:59 9999

ctime: Sat Jan 1 00:00:00 10000

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

时间和语言 > 日期和时间

11:14

2038年1月19日



时区

(UTC+08:00) 北京, 重庆, 香港特别行政区, 乌鲁木齐

时间和语言 > 日期和时间

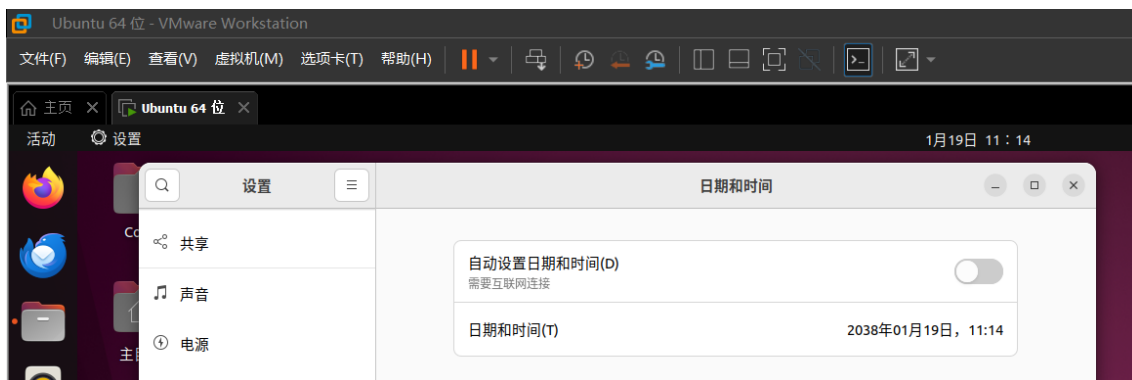
11:15

2038年1月19日



时区

(UTC+08:00) 北京, 重庆, 香港特别行政区,



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）。
编写程序在内存验证手动编码的正确性，截图。

浮点数：-10.1

符号位：1

整数部分二进制为：0b1010

小数部分二进制为：0.000110011001100110011001100...

二者结合为：1010.00011001100110011001100...

写为规格数有：1.010 0001 1001 1001 1001 1001 1001 [1001]... * 2³

向偶数取整为：1.010 0001 1001 1001 1001 1010 * 2³

故我们有：

符号位：1

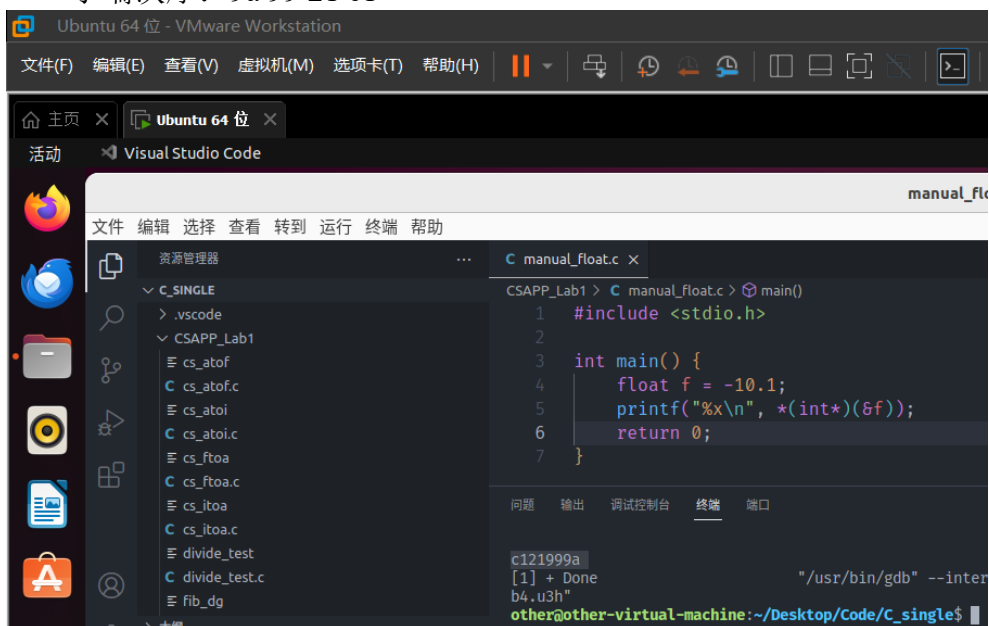
阶码：3+127=130，即 1000 0010

尾码：010 0001 1001 1001 1001 1010

组合起来为：1100 0001 0010 0001 1001 1001 1001 1010

转换为十六进制：c1 21 99 9a

小端次序：9a 99 21 c1



```
Ubuntu 64 位 - VMWare Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
活动 Visual Studio Code
手动_float
文件 编辑 选择 查看 转到 运行 终端 帮助
资源管理器
C_SINGLE
> .vscode
> CSAPP_Lab1
  cs_atof
  cs_atof.c
  cs_atoi
  cs_atoi.c
  cs_ftoa
  cs_ftoa.c
  cs_itoa
  cs_itoa.c
  divide_test
  divide_test.c
  fib_dg
  fib_dg.c
C manual_float.c x
CSAPP_Lab1 > C manual_float.c > main()
1 #include <stdio.h>
2
3 int main() {
4     float f = -10.1;
5     printf("%x\n", *(int*)&f);
6     return 0;
7 }
问题 输出 调试控制台 终端 端口
c121999a
[1] + Done "/usr/bin/gdb" --inter
b4.u3h"
other@other-virtual-machine:~/Desktop/Code/C_single$
```

7.2 特殊 float 数据的处理

```
+0:      0.00000000000000000000000000000000000000000000000000000000000000e+00  
        00000000  
  
-0:     -0.00000000000000000000000000000000000000000000000000000000000000e+00  
        80000000  
  
最小浮点正数:  
    1.40129846432481707092372958328991613128026194187651577175706828388979108268586060148663818836212158203125000000000000000e-45  
    00000001  
  
最大浮点正数:  
    3.40282346638528859811704183484516925440000000000000000000000000000000000000000000000000000000e+38  
    7fffffffff  
  
最小正的规格化浮点数:  
    1.17549435082287507968736537222456778186655567720875215087517062784172594547271728515625000000000000000000000000e-38  
    00800000  
  
正无穷大:  
    inf  
    7f800000  
  
NaN:  
    nan  
    7fc00000
```

提交子程序 float0.c

```

inf
inf
[1] + Done                                "/usr/bin/gdb" --in
1.mcq"
other@other-virtual-machine:~/Desktop/Code/C_single$

```

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？

7.5 讨论 1: 有多少个 int 可以用 float 精确表示

有 150994944 个 int 数据可以用 float 精确表示。

```

#include <stdio.h>
#include <limits.h>

int main() {
    int x = INT_MIN, n = 0;
    while (x != INT_MAX) {
        if ((double)x == (double)(float)x)
            ++n;
        ++x;
    }
    if ((double)x == (double)(float)x)
        ++n;
    printf("%d\n", n);
    return 0;
}

```

Microsoft Visual Studio 调试

150994944

D:\VS2022\CSAPP_Lab1_1\x64\Debug\CSAPP_Lab1_1.exe

是哪些数据呢？0，正负(1~2²⁴-1)，正负(不在先前范围，对二进制，除去后导'0'后，最高的'1'的位置不大于 23 位)

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！

```

void show_float(float f) {
    printf("%08x\n", *(int*)&f);
}

int main() {
    float a = 0b100000000000000000000001;
    //      ^      ^      ^      ^
    //      1      10     20     25
    float b = 0b1000000000000000000000011;
    //      ^      ^      ^      ^
    //      1      10     20     25

    show_float(a);
    show_float(b);

    return 0;
}

```

Microsoft Visual

4b800000
4b800002

D:\VS2022\CSAPP
按任意键关闭此窗

图中 a 被 1 000 0000 0000 0000 0000 0000 1 赋值，最后一位需进位
由于 1 正好是中间值，向偶数舍入，前一位已经是 0，故不进位
结果为 1 000 0000 0000 0000 0000 0000，尾码全为 0
与图中 4b800000 相符

而图中 b 被 1 000 0000 0000 0000 0000 0001 1 赋值，最后一位需进位
 由于 1 正好是中间值，向偶数舍入，前一位是 1，应进位
 结果为 1 000 0000 0000 0000 0000 0010，尾码为 0x000002(后 23 位)
 与图中 4b800002 相符

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, 0.5, 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): $0, \pm[2^{-149} \sim (1-2^{-23}) \times 2^{-126}]$ 、 2^{-149} 、 $\pm[2^{127} \sim (2-2^{-23}) \times 2^{127}]$ 、 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 $1.40129846432e-45$

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 $2.02824120215e+31$

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。

因为浮点数不精确的特性，两个浮点数使用 == 直接比较经常不能达到我们的预期目标

判断两个浮点数是否相等，一个比较好的方法就是用两个数差值的绝对值小于某个极小值，如 $1e-5$ ，来判断，即

$$\text{fabs}(a-b) \leq 1e-6$$

以此类推，判断大于为 $a > b \ \&\& \ \text{fabs}(a-b) > 1e-6$

判断小于为 $a < b \ \&\& \ \text{fabs}(a-b) > 1e-6$

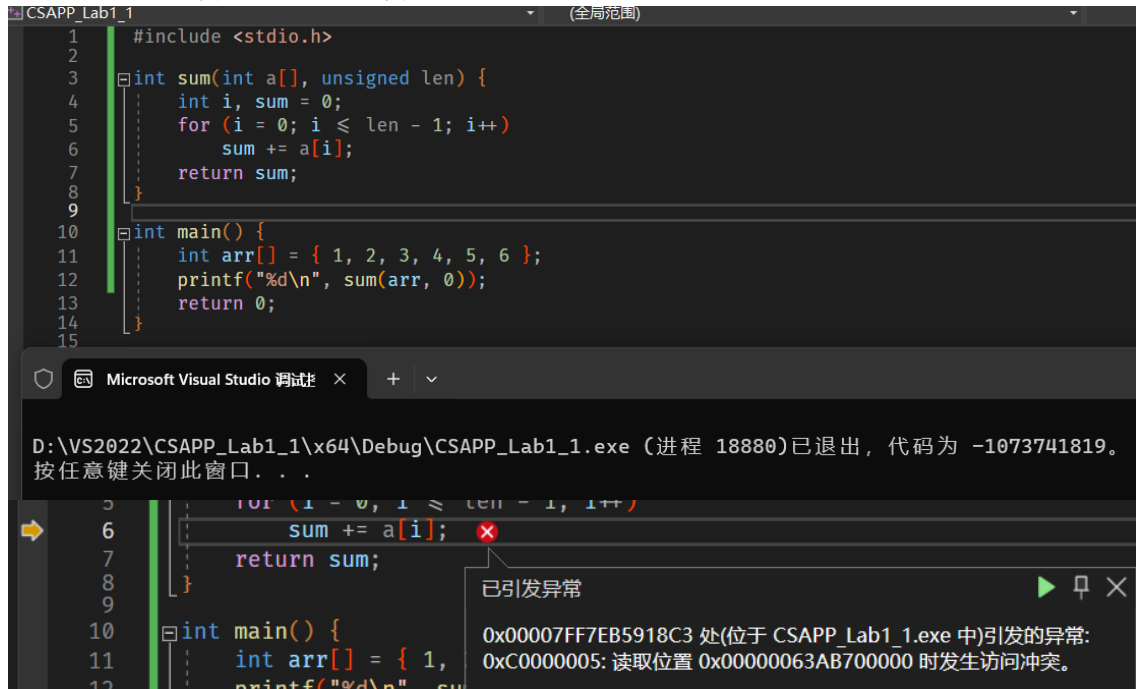
如果要手动进行两个 IEEE754 标准的浮点数之间的比较：

- 1、判断特殊数 (inf 与 nan)，对于特殊数之间比较的结果需特殊指定
- 2、判断符号位，如果一个为正，一个为负，则正数大于负数。若为 0，则正 0 和负 0 相等
- 3、比较阶码，若二者符号位为正，则阶码大的数大；符号位为负，则解码小的数大
- 4、比较尾数，若二者符号位为正，尾数大的大；符号位为负，则尾数小的的大

第 8 章 程序运行分析

8.1 隐式类型转换

1.截图说明运行结果，并原因分析。



```
1 #include <stdio.h>
2
3 int sum(int a[], unsigned len) {
4     int i, sum = 0;
5     for (i = 0; i <= len - 1; i++)
6         sum += a[i];
7     return sum;
8 }
9
10 int main() {
11     int arr[] = { 1, 2, 3, 4, 5, 6 };
12     printf("%d\n", sum(arr, 0));
13     return 0;
14 }
15
```

D:\VS2022\CSAPP_Lab1_1\x64\Debug\CSAPP_Lab1_1.exe (进程 18880) 已退出，代码为 -1073741819。
按任意键关闭此窗口。...

已引发异常
0x00007FF7EB5918C3 处(位于 CSAPP_Lab1_1.exe 中)引发的异常:
0xC0000005: 读取位置 0x00000063AB700000 时发生访问冲突。

进入 sum 函数体时，len=0

在计算 $i \leq \text{len} - 1$ 时：

len : 0x00000000

1: 0x00000001

len - 1 = 0xffffffff

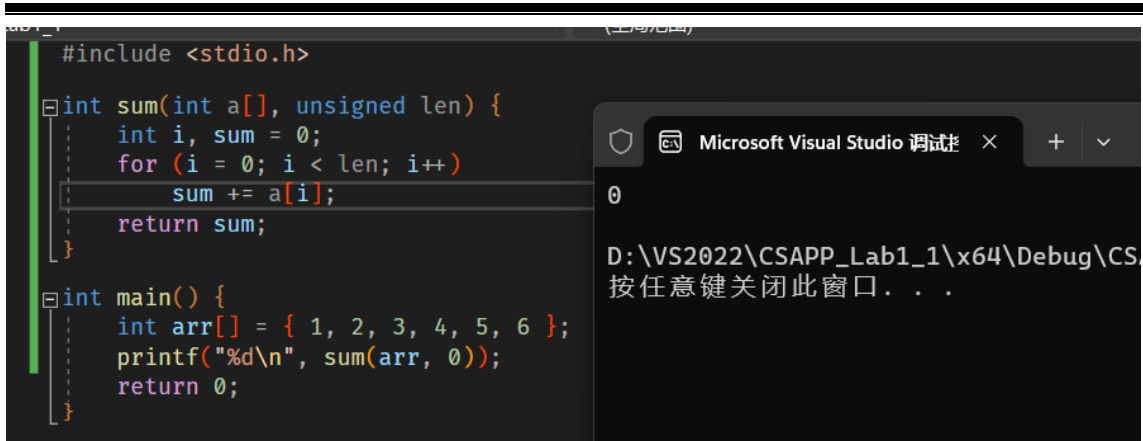
由于 len 为无符号数，结果 0xffffffff 将按无符号数解释，为无符号数的最大值

故对任意的 i，均有 $i \leq \text{len} - 1$ 为真，该 for 循环为死循环，i 将不受控制地增大

当 i 的值超过输入数组 a 的长度时，a[i] 将越界访问，产生未定义行为

2.论述改进方法

法一：修改 sum 中 for 循环的继续条件



```
#include <stdio.h>

int sum(int a[], unsigned len) {
    int i, sum = 0;
    for (i = 0; i < len; i++)
        sum += a[i];
    return sum;
}

int main() {
    int arr[] = { 1, 2, 3, 4, 5, 6 };
    printf("%d\n", sum(arr, 0));
    return 0;
}
```

Microsoft Visual Studio 调试

0

D:\VS2022\CSAPP_Lab1_1\x64\Debug\CSAPP_Lab1_1.exe (进程 23104) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

法二：修改 sum 中形参 len 的数据类型



```
#include <stdio.h>

int sum(int a[], int len) {
    int i, sum = 0;
    for (i = 0; i ≤ len - 1; i++)
        sum += a[i];
    return sum;
}

int main() {
    int arr[] = { 1, 2, 3, 4, 5, 6 };
    printf("%d\n", sum(arr, 0));
    return 0;
}
```


Microsoft Visual Studio 调试

0

D:\VS2022\CSAPP_Lab1_1\x64\Debug\CSAPP_Lab1_1.exe (进程 23104) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

8.2 浮点数的精度

1. 运行结果截图，分析产生原因。



```
请输入一个浮点数:61.419997
这个浮点数的值是:61.419998
请输入一个浮点数:61.419998
这个浮点数的值是:61.419998
请输入一个浮点数:61.419999
这个浮点数的值是:61.419998
请输入一个浮点数:61.420000
这个浮点数的值是:61.419998
请输入一个浮点数:61.420001
这个浮点数的值是:61.420002
请输入一个浮点数:0
这个浮点数的值是:0.000000

D:\VS2022\CSAPP_Lab1_1\x64\Debug\CSAPP_Lab1_1.exe (进程 23104) 已退出，代码为 0。  
按任意键关闭此窗口。 . . .
```

```

请输入一个浮点数:10.186810
这个浮点数的值是:10.186810
请输入一个浮点数:10.186811
这个浮点数的值是:10.186811
请输入一个浮点数:10.186812
这个浮点数的值是:10.186812
请输入一个浮点数:10.186813
这个浮点数的值是:10.186813
请输入一个浮点数:10.186814
这个浮点数的值是:10.186814
请输入一个浮点数:10.186815
这个浮点数的值是:10.186815
请输入一个浮点数:0
这个浮点数的值是:0.000000

```

D:\VS2022\CSAPP_Lab1_1\x64\Debug\CSAPP_Lab1_1.exe (进程 20868)已退出, 代码为 0。
按任意键关闭此窗口...

61.xxxx 大约写成二进制科学计数法为: $1.xxxxx \times 2^5$

当指数为 5 时, float 最小增量为: $2^5 \times 2^{-23} = 2^{-18}$, 约为 $3.8e-5$

而我们的输入 61.419997~61.420002 的增量为 $1e-6$, 小于最小增量, 故会出现不同的输入对应同样的输出的情况

而 10.xxxx 大约写成二进制科学计数法为: $1.xxxxx \times 2^3$

当指数为 3 时, float 最小增量为: $2^3 \times 2^{-23} = 2^{-20}$, 约为 $9.5e-7$

我们的输入 10.186810~10.186815 的增量为 $1e-6$, 大于最小增量, 故输出可达到我们的预期

2. 论述编程中浮点数比较、汇总统计等应如何正确编程。

1、避免直接比较浮点数。当判断两个浮点数是否相同时, 不能简单粗暴的使用 $a == b$ 来判断, 可采用绝对误差 ε , 即当 $|a - b| < \varepsilon$ 时认为二者相等; 或采用相

对误差 δ , 当 $\left| \frac{a-b}{a} \right| < \delta$ 时认为二者相等。

2、对于浮点数的特殊值 `inf` 与 `nan` 应小心处理

3、当进行一些统计操作, 如取平均值时, 应先进行加减法, 再进行除法, 减少浮点数的出现次数

4、在允许的情况下使用精度更高的 `double` 而不是 `float`

第 9 章 舍尾平衡的讨论

9.1 描述可能出现的问题

在数据处理过程中，有时需要进行舍位平衡操作。舍入处理经常使用四舍五入计算方法，但这可能引入误差。如果在统计中存在对这些数据的总和计算，舍入产生的误差就会逐渐累积，可能导致舍入后的数据与其总和不匹配。

例如： $1.005+1.005=2.01$ ，但舍入后 $1.01+1.01=2.02$ ，与原数据有出入

9.2 给出完美的解决方案

- a、在运算时提高精度，即 `float` \rightarrow `double` \rightarrow `long double`
- b、在数据传递、上报时提高精度，即：对于要求精度为整数的值，应至少上报 2 位小数等
- c、抛弃浮点数，对于一些精度要求极高的数，使用高精度数来进行运算及传输
- d、在计算时，通过交换运算次序等方法，尽量减少误差的产生

第 10 章 总结

10.1 请总结本次实验的收获

通过本次实验，本人巩固了在 linux 下的操作，了解了在 VS 以及 Linux 下的反编译操作，知道了常用编码与汉字编码的原理，了解了整数的溢出所可能带来的危害，学习了 ieee754 标准中浮点数的表示方法及其缺陷，以及我们应该如何更加安全、正确地使用浮点数进行运算

10.2 请给出对本次实验内容的建议

可以在某些问题下给予一定的提示

注：本章为酌情加分项。

参考文献

- [1] ubuntu 上运行 codeblocks 中文乱码
<https://blog.csdn.net/yangtuanzi1118/article/details/79585386>
- [2] ubuntu 64 上的 GCC 如何编译 32 位程序
<https://blog.csdn.net/longintchar/article/details/50557832>
- [3] objdump(Linux)反汇编命令使用指南
<https://blog.csdn.net/wwchao2012/article/details/79980514>
- [4] 编译器警告（级别 3）C4996
<https://learn.microsoft.com/zh-cn/cpp/error-messages/compiler-warnings/compiler-warning-level-3-c4996>
- [5] IEEE Standard 754 Floating Point Numbers
<https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/>
- [6] IEEE 754 Converter (JavaScript), V0.22
<https://www.h-schmidt.net/FloatConverter/IEEE754.html>