

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
学 号	1190201423
班 级	1903004
学 生	顾海耀
指 导 教 师	史先俊
实 验 地 点	G709
实 验 日 期	2021/3/31

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	- 8 -
2.1 UBUNTU 下 CODEBLOCKS 安装	- 8 -
2.2 64 位 UBUNTU 下 32 位运行环境建立	- 8 -
第 3 章 C 语言的数据类型与存储	- 10 -
3.1 类型本质	- 10 -
3.2 数据的位置-地址	- 10 -
3.3 MAIN 的参数分析	- 12 -
3.4 指针与字符串的区别	- 14 -
第 4 章 深入分析 UTF-8 编码	- 16 -
4.1 提交 UTF8LEN.C 子程序	- 16 -
4.2 C 语言的 STRCMP 函数分析	- 16 -
4.3 讨论：按照姓氏笔画排序的方法实现	- 16 -
第 5 章 数据变换与输入输出	- 17 -
5.1 提交 CS_ATOI.C	- 17 -
5.2 提交 CS_ATOF.C	- 17 -
5.3 提交 CS_ITOA.C	- 17 -
5.4 提交 CS_FTOA.C	- 17 -
5.5 讨论分析 OS 的函数对输入输出的数据有类型要求吗	- 17 -
第 6 章 整数表示与运算	- 18 -
6.1 提交 FIB_DG.C	- 18 -
6.2 提交 FIB_LOOP.C	- 18 -
6.3 FIB 溢出验证	- 18 -

6.4 除以 0 验证:	- 18 -
6.5 万年虫验证.....	- 19 -
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.....	- 23 -
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 舍尾平衡的讨论	- 26 -
8.1 描述可能出现的问题.....	- 26 -
8.2 给出完美的解决方案.....	- 26 -
第 9 章 总结.....	- 27 -
9.1 请总结本次实验的收获.....	- 27 -
9.2 请给出对本次实验内容的建议.....	- 27 -
参考文献.....	- 28 -

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

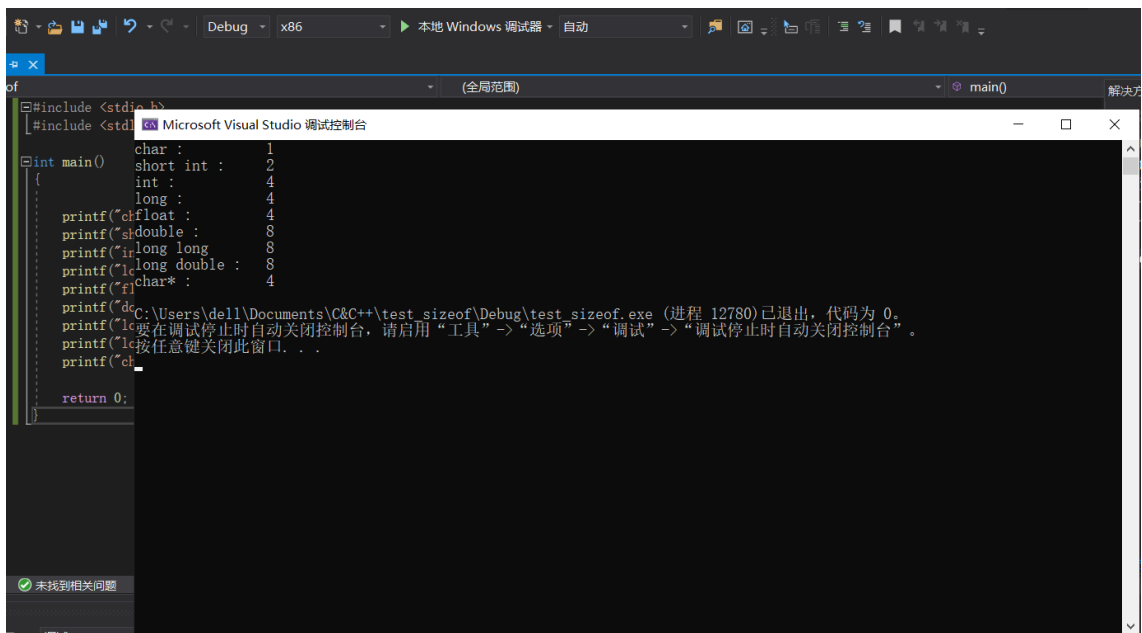
1.3 实验预习

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。

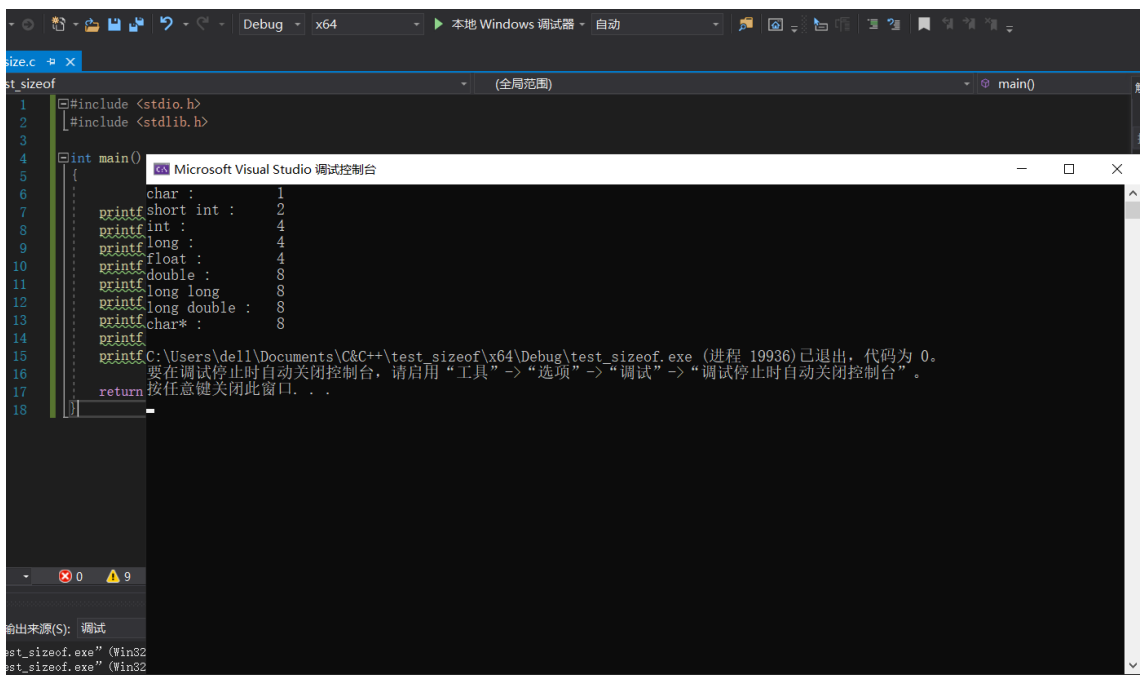
采用 `sizeof` 在 Windows 的 VS/CB 以及 Linux 的 CB/GCC 下获得 C 语言每一类型在 32/64 位模式下的空间大小

计算机系统实验报告



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

int main()
{
    char : 1
    short int : 2
    int : 4
    long : 4
    printf("sizeof : 4\n");
    printf("%d\n", sizeof double);
    printf("%s\n", "long long");
    printf("%c\n", 'long double');
    printf("%f\n", char*);
    printf("C:\Users\dell\Documents\C&C++\test_sizeof\Debug\test_sizeof.exe (进程 12780) 已退出, 代码为 0。");
    printf("要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。");
    printf("按任意键关闭此窗口。");
    return 0;
}
```



```
size.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     char : 1
7     printf short int : 2
8     printf int : 4
9     printf long : 4
10    printf float : 4
11    printf double : 8
12    printf long long : 8
13    printf long double : 8
14    printf char* : 8
15    printf "C:\Users\dell\Documents\C&C++\test_sizeof\x64\Debug\test_sizeof.exe (进程 19936) 已退出, 代码为 0。");
16    printf "要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。");
17    printf "按任意键关闭此窗口。");
18    return 0;
19 }
```

```

ghy1190201423@ubuntu: /mnt/hgfs/hitcs
File Edit View Search Terminal Help
ghy1190201423@ubuntu:/mnt/hgfs/hitcs$ vim test_size.c
ghy1190201423@ubuntu:/mnt/hgfs/hitcs$ gcc -m32 -Og test_size.c -o test_size32
ghy1190201423@ubuntu:/mnt/hgfs/hitcs$ ./test_size32
char :      1
short int :  2
int :       4
long :      4
float :      4
double :     8
long long   8
long double : 12
char* :      4
ghy1190201423@ubuntu:/mnt/hgfs/hitcs$ ./test_size64
char :      1
short int :  2
int :       4
long :      8
float :      4
double :     8
long long   8
long double : 16
char* :      8
ghy1190201423@ubuntu:/mnt/hgfs/hitcs$
ghy1190201423@ubuntu:/mnt/hgfs/hitcs$

```

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）

int:47

long:47

unsigned int:48

unsigned long:48

先用递归程序实现，会出现什么问题？递归会非常慢

再用循环方式实现。

递归时间复杂度： $O(2^n)$ 循环时间复杂度： $O(n)$

写出 float/double 类型最小的正数、最大的正数（非无穷）

float 类型最小正数： 2^{-149} (00000000000000000000000000000001)

float 类型最大正数： $(2^{128}-2^{104})$ (01111111011111111111111111111111)

double类型最小正数: 2^{-1074}

double类型最大正数: $(2^{1024}-2^{971})$

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

9a 99 21 c1

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)

$\frac{-(2^{-126}-2^{-149}) \sim (2^{-126}-2^{-149})}{2^{-149}}$ 、 $\frac{-(2^{127}-2^{104}) \sim -2^{127}}{2^{104}}$ 和 $\frac{2^{127} \sim (2^{127}-2^{104})}{2^{104}}$

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

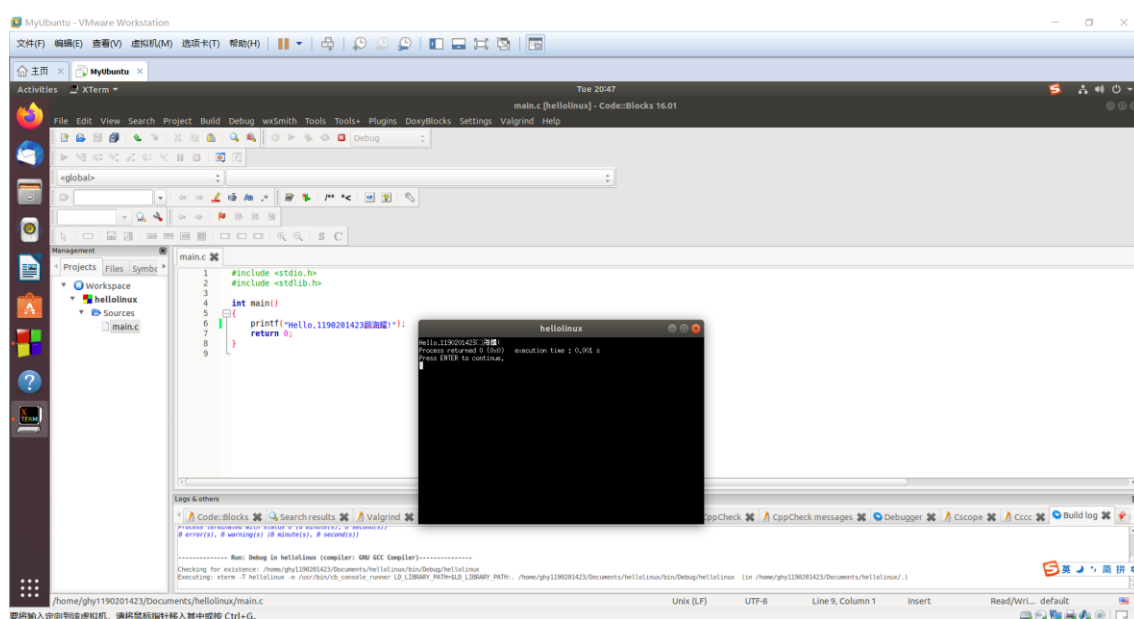
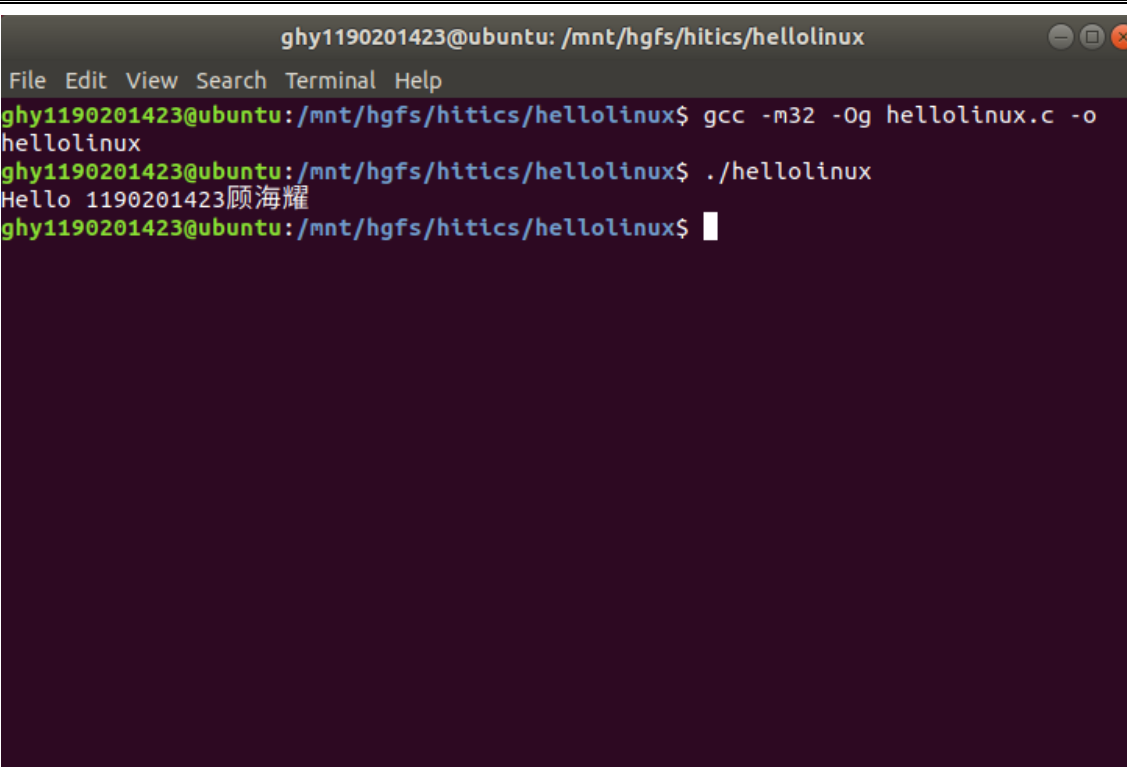


图 2-1 Ubuntu 下 CodeBlocks 截图

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。



```
ghy1190201423@ubuntu: /mnt/hgfs/hitics/hellolinux
File Edit View Search Terminal Help
ghy1190201423@ubuntu:/mnt/hgfs/hitics/hellolinux$ gcc -m32 -Og hellolinux.c -o
hellolinux
ghy1190201423@ubuntu:/mnt/hgfs/hitics/hellolinux$ ./hellolinux
Hello 1190201423 顾海耀
ghy1190201423@ubuntu:/mnt/hgfs/hitics/hellolinux$
```

图 2-2 Ubuntu 与 Windows 共享目录截图

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/32	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 不是函数,而是运算符,C 语言编译器在预编译阶段的时候就已经处理完了 sizeof 的问题,也就是说 sizeof 类似于宏定义。

3.2 数据的位置-地址

打印 x、y、z 输出的值：截图 1

反汇编查看 x、y、z 的地址，每字节的内容：截图 2，标注说明

反汇编查看 x、y、z 在代码段的表示形式。截图 3，标注说明

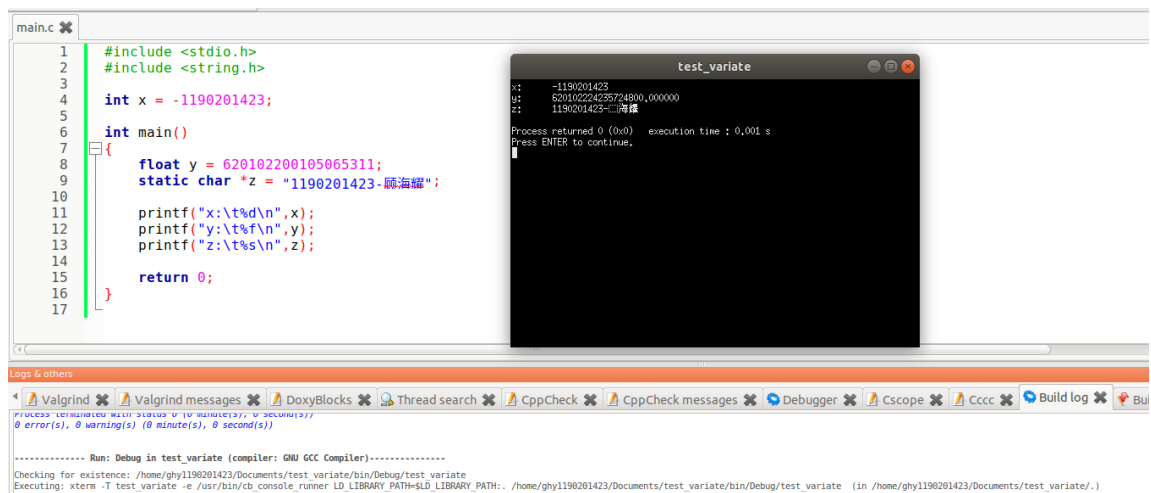
x 与 y 在汇编阶段转换成补码与 ieee754 编码。

数值型常量与变量在存储空间上的区别是：宏常量不存储，常量在代码段，静态变量和全局变量在数据段，局部变量在堆栈段。

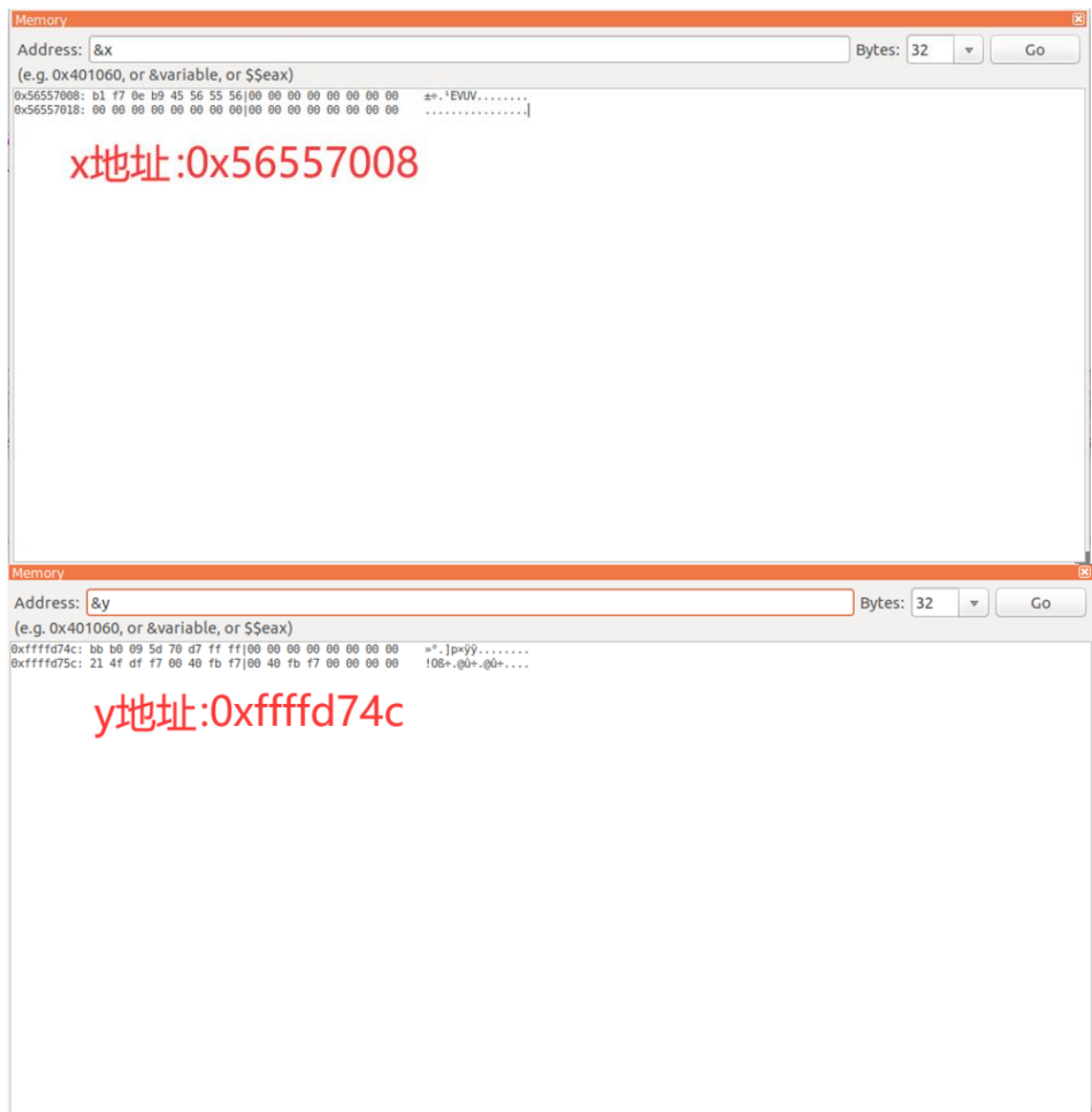
字符串常量与变量在存储空间上的区别是：字符串常量在代码段，静态变量和全局变量在数据段，局部变量在堆栈段。

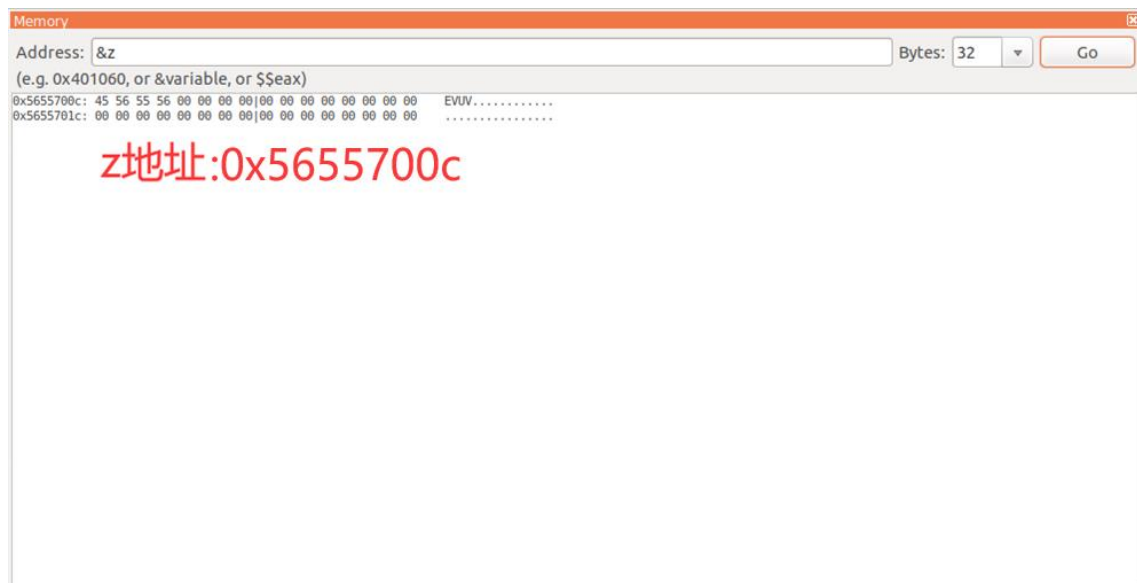
常量表达式在计算机中处理方法是：在编译阶段转换成常量来存储。

计算机系统实验报告



截图 1





截图 2

```

11      printf("x:\t%d\n",x);
0x56555553  mov     eax,DWORD PTR [ebx+0x30]
0x56555559  sub     esp,0x8
0x5655555c  push    eax
0x5655555d  lea     eax,[ebx-0x19a8] X
0x56555563  push    eax
0x56555564  call    0x565553c0 <printf@plt>
0x56555569  add     esp,0x10
12      printf("y:\t%f\n",y);
0x5655556c  fld     DWORD PTR [ebp-0xc]
0x5655556f  sub     esp,0x4
0x56555572  lea     esp,[esp-0x8]
0x56555576  fstp    QWORD PTR [esp]
0x56555579  lea     eax,[ebx-0x19a1] Y
0x5655557f  push    eax
0x56555580  call    0x565553c0 <printf@plt>
0x56555585  add     esp,0x10
13      printf("z:\t%s\n",z);
0x56555588  mov     eax,DWORD PTR [ebx+0x34]
0x5655558e  sub     esp,0x8
0x56555591  push    eax
0x56555592  lea     eax,[ebx-0x199a] Z
0x56555598  push    eax
0x56555599  call    0x565553c0 <printf@plt>
0x5655559e  add     esp,0x10

```

截图 3

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 地址见截图 3

Memory

Address:
Bytes:

(e.g. 0x401060, or &variable, or \$\$eax)

```

0xffffd770: 01 00 00 00 04 d8 ff ff|0c d8 ff ff 94 d7 ff ff  ....0yy.0yy[0x24]xÿÿ
0xffffd780: 01 00 00 00 04 d8 ff ff|00 40 fb f7 1a 57 fe f7  ....0yy.@û+.Wp+

```

argc 地址:0xffffd770

Memory

Address:
Bytes:

(e.g. 0x401060, or &variable, or \$\$eax)

```

0xffffd774: 04 d8 ff ff 0c d8 ff ff|94 d7 ff ff 01 00 00 00  .0yy.0yy[0x24]xÿÿ....
0xffffd784: 04 d8 ff ff 00 40 fb f7|1a 57 fe f7 00 d8 ff ff  .0yy.@û+.Wp+.0yy

```

argv地址: 0xffffd774

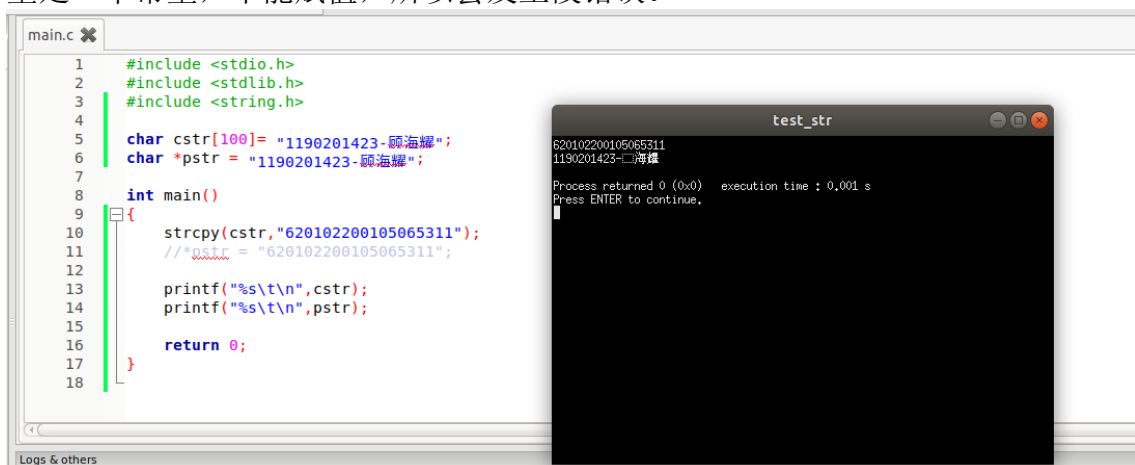


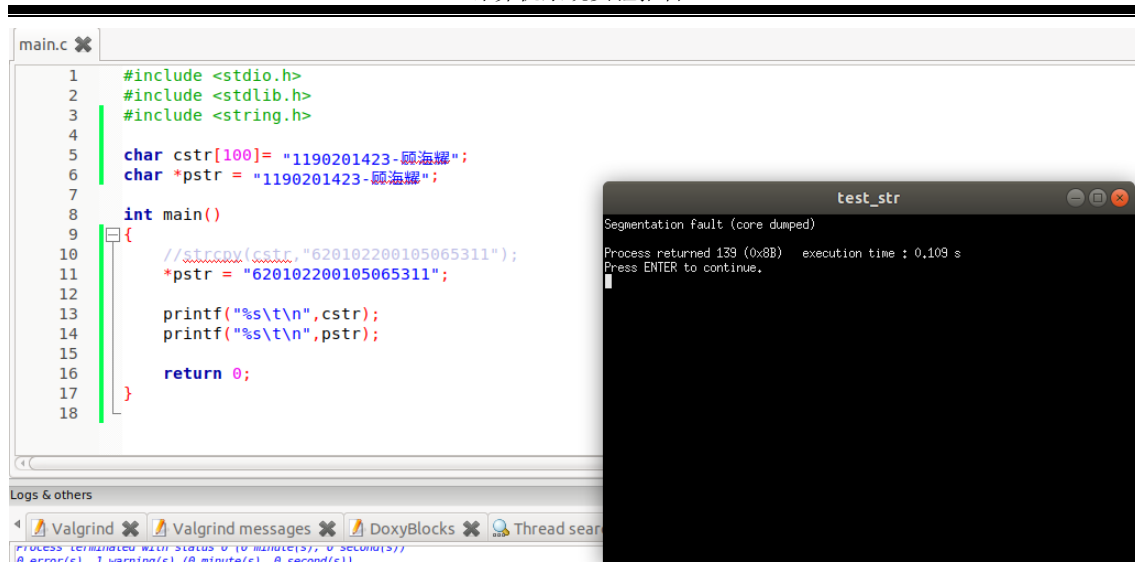
截图 4

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

pstr 修改内容会出现什么问题：因为 pstr 是一个全局指针变量，而全局指针变量是一个常量，不能赋值，所以会发生段错误。





截图 5

第 4 章 深入分析 UTF-8 编码

4.1 提交 utf8len.c 子程序

见附件 utf8len.c

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

因为 strcmp 函数是根据 ASCII 字符集进行比较的，而在 ASCII 字符集中并没有汉字，但是 unicode 字符集中是有汉字的，所以 strcmp 先将汉字转换为 unicode 编码，然后根据 unicode 编码规则进行排序。

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？

我认为，首先使用数据库存储姓氏笔画数量的相关信息，排序时，首先根据**笔画的数量**进行排序，若笔画数量相同，可以根据**起笔的笔顺**进行排序，比如可以根据“横、竖、撇、点、折、其他”的顺序进行排序，如果相同，继续比较接下来的笔顺，若一方已经结束就排小于。

第 5 章 数据变换与输入输出

5.1 提交 `cs_atoi.c`

见附件 `cs_atoi.c`

5.2 提交 `cs_atof.c`

见附件 `cs_atof.c`

5.3 提交 `cs_itoa.c`

见附件 `cs_itoa.c`

5.4 提交 `cs_ftoa.c`

见附件 `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：

我认为有要求：

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，其中两个函数为：

`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：若成功则为读的字节数，若出错则为-1]

`read` 函数有一个 `size_t` 的输入参数和一个 `ssize_t` 的返回值。在 x86-64 系统中，`size_t` 被定义为 `unsigned long`，而 `ssize_t` (有符号的大小) 被定义为 `long`。`read` 函数返回一个有符号的大小，而不是一个无符号大小，这是因为出错时它必须返回-1。`Write` 函数从描述符为 `buf` 的当前文件位置复制最多 `n` 个字节到描述符 `buf` 的当前位置。

第 6 章 整数表示与运算

6.1 提交 fib_dg.c

见附件 fib_dg.c

6.2 提交 fib_loop.c

见附件 fib_loop.c

6.3 fib 溢出验证

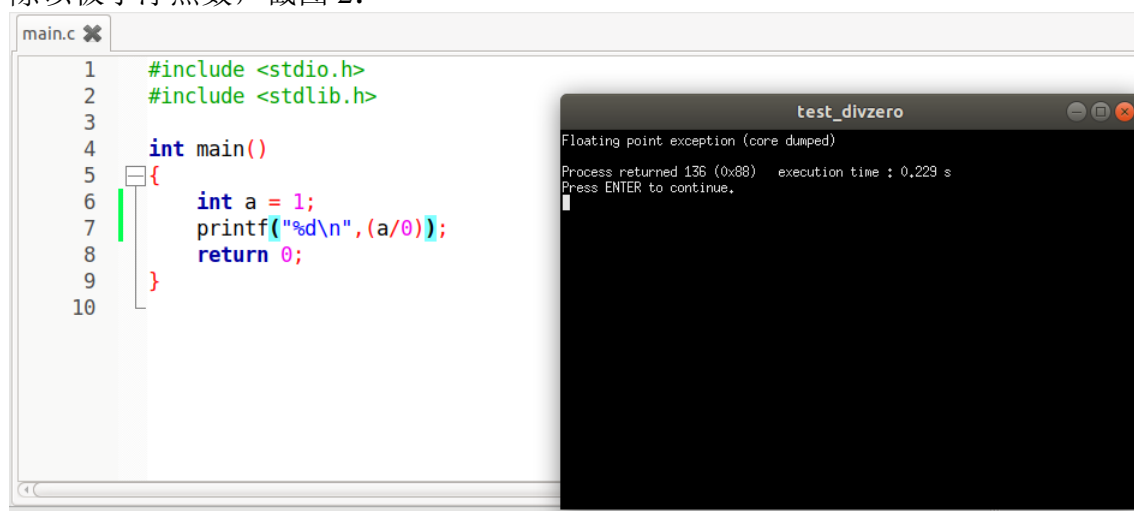
int 时从 n=47 时溢出, long 时 n=47 时溢出。

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

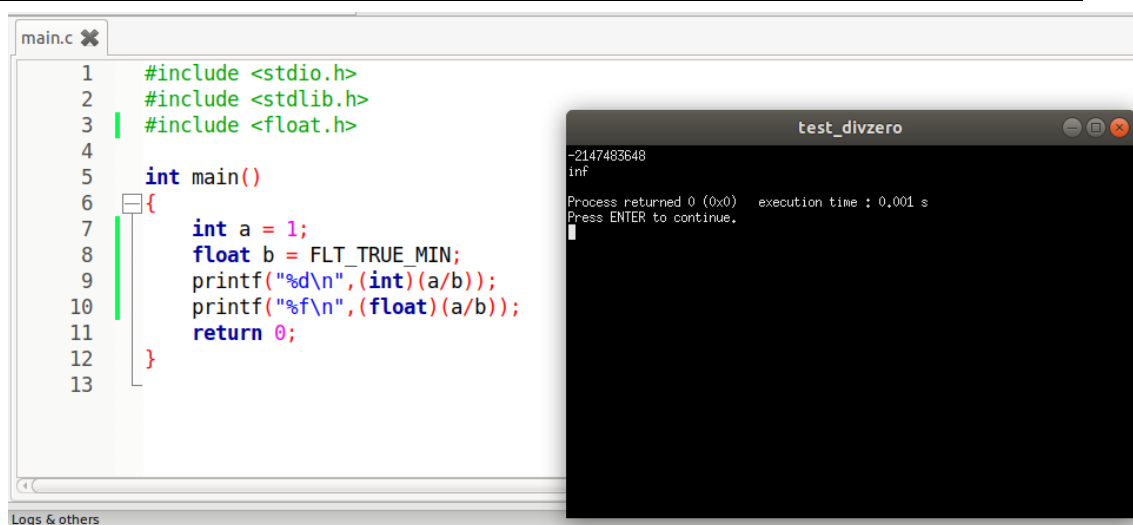
6.4 除以 0 验证:

除以 0: 截图 1

除以极小浮点数, 截图 2:



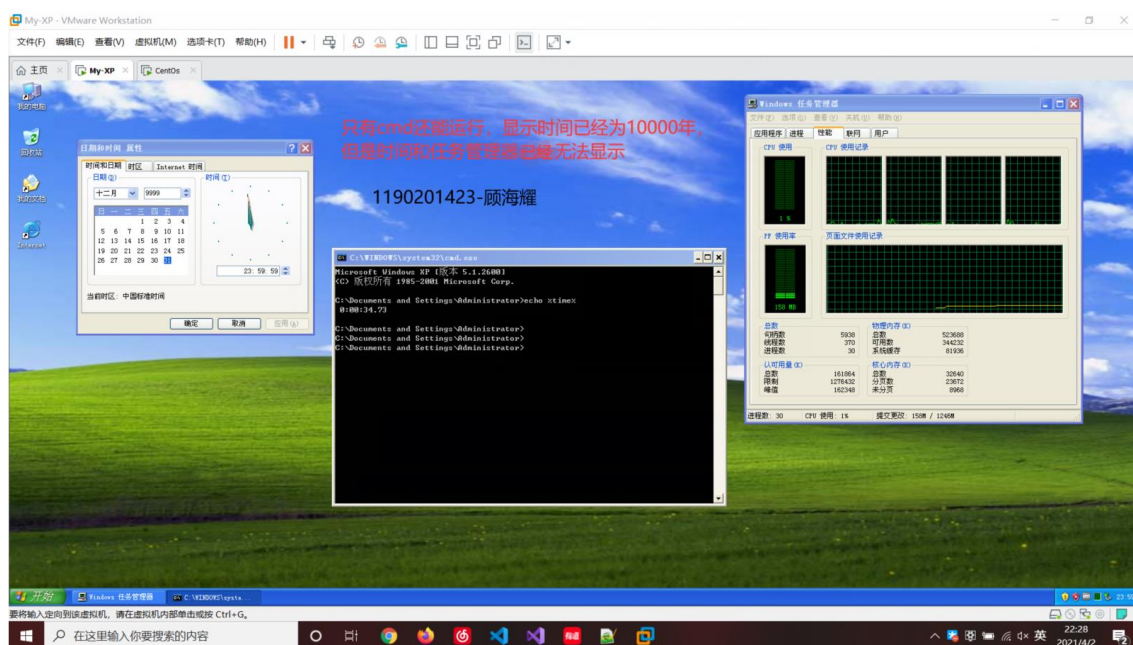
截图 1



截图 2

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:

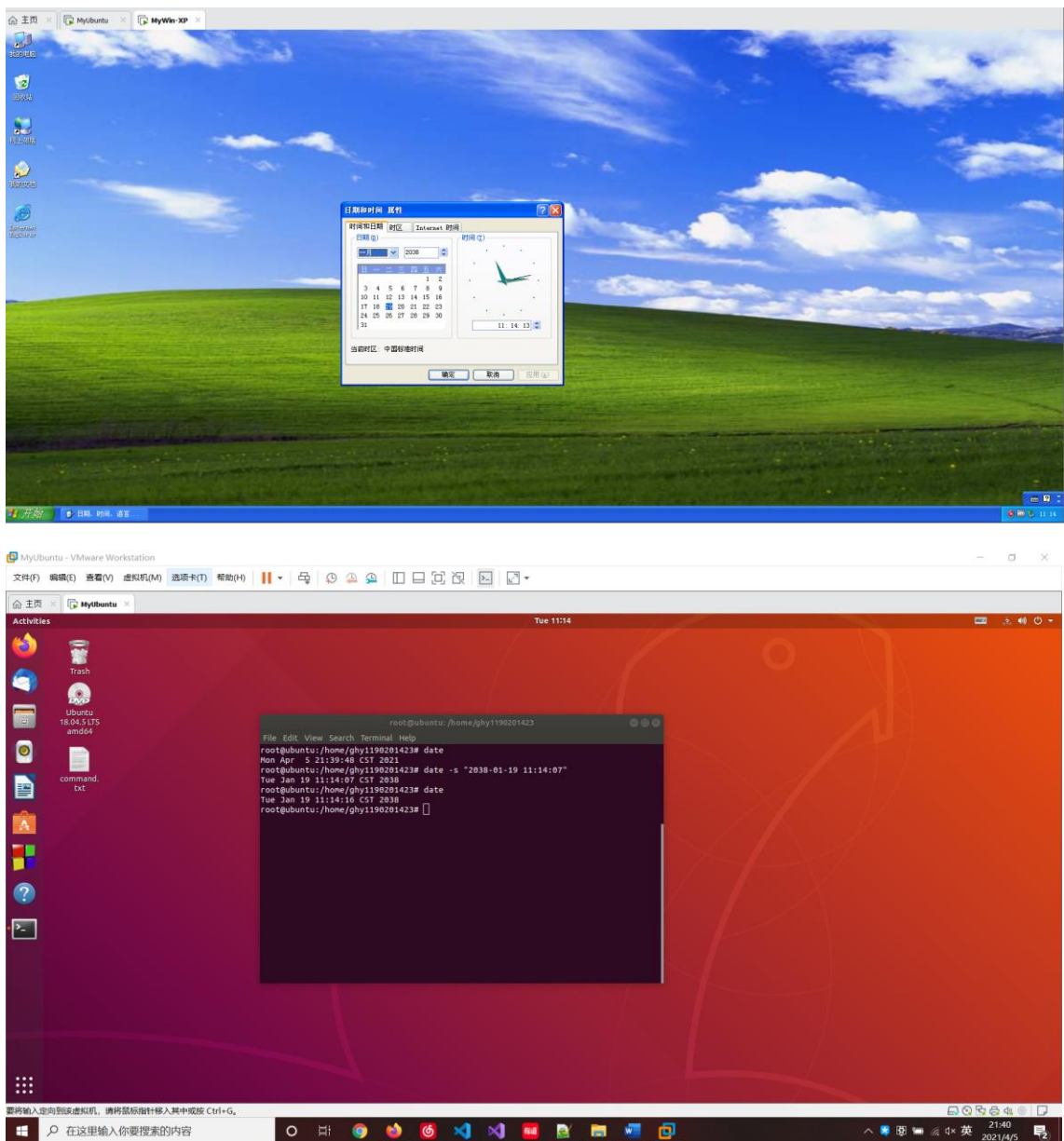


Linux 调整到 9999 年 12 月 31 日会报错: “Invalid argument”。

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

计算机系统实验报告



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。

首先-10.1写成二进制为 $-1010.00011[0011] \dots = -1.01000011[0011] \dots \times 2^3$

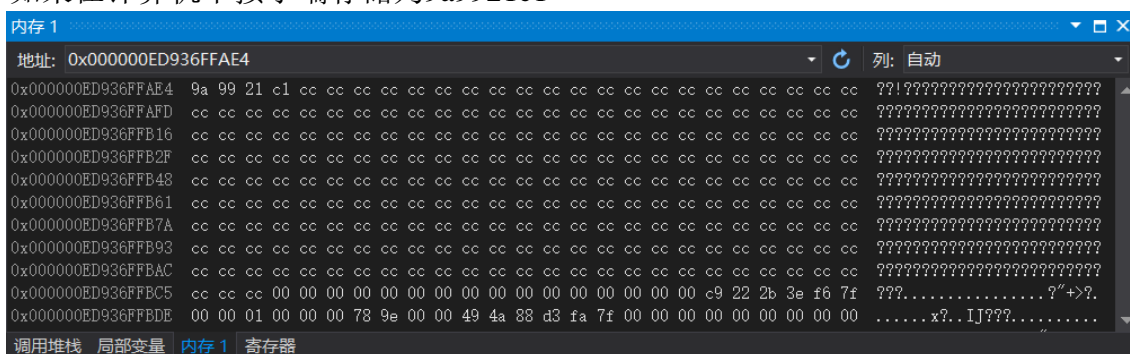
然后首先符号位为1

接下来确定阶码， $3+127_{10}=130_{10}=10000010_2$

最后看尾数01000011001100110011010

最后就可以得出结果11000001001000011001100110011010

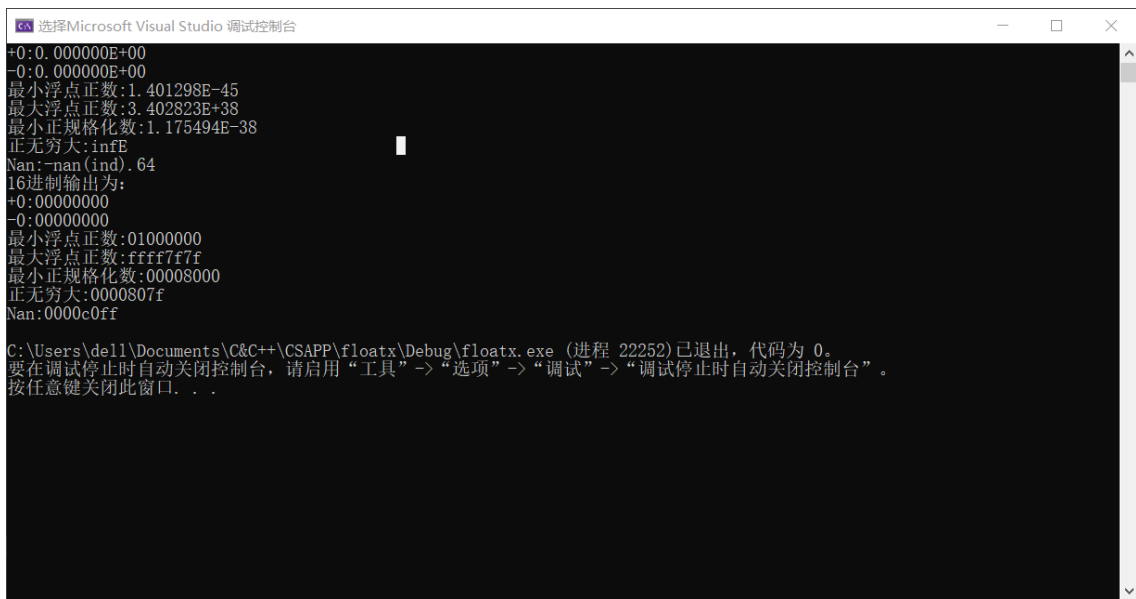
如果在计算机中按小端存储为9a9921c1



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。



```

选择Microsoft Visual Studio 调试控制台
+0: 0.000000E+00
-0: 0.000000E+00
最小浮点正数: 1.401298E-45
最大浮点正数: 3.402823E+38
最小正规格化数: 1.175494E-38
正无穷大: infE
Nan: -nan(ind).64
16进制输出为:
+0: 00000000
-0: 00000000
最小浮点正数: 01000000
最大浮点正数: ffff7fff
最小正规格化数: 00008000
正无穷大: 0000807f
Nan: 0000c0ff

C:\Users\dell\Documents\C&C++\CSAPP\floatx\Debug\floatx.exe (进程 22252) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。

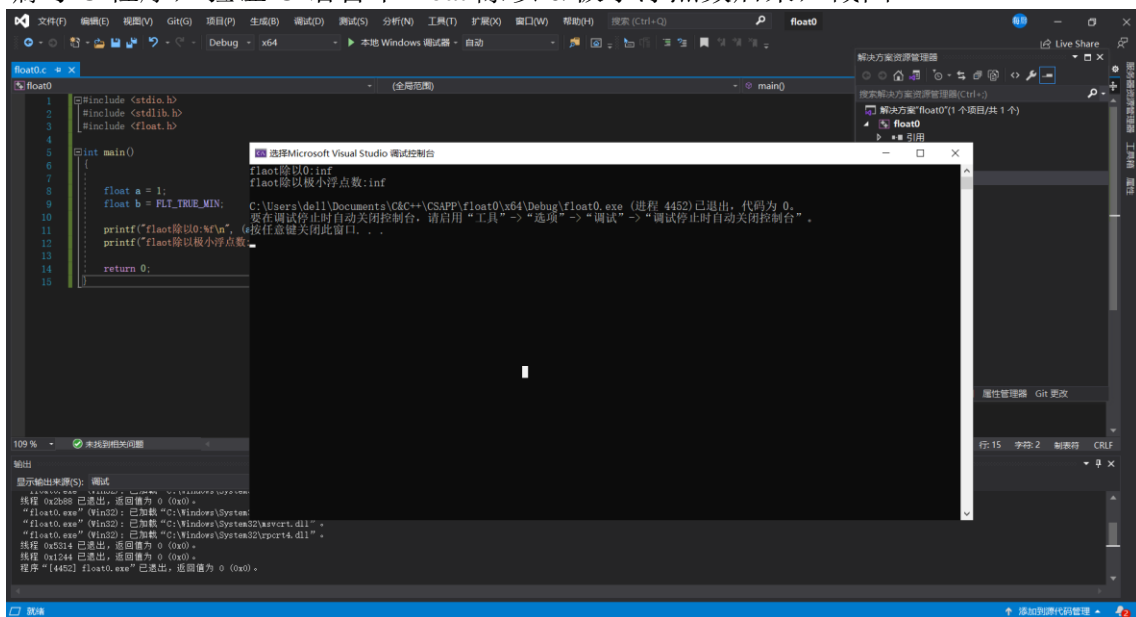
```

见附件 floatx.c

7.3 验证浮点运算的溢出

提交子程序 float0.c

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



```

float0.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <float.h>
4
5 int main()
6 {
7     float a = 1;
8     float b = FLT_MIN;
9
10    printf("float除以0:\n");
11    printf("float除以极小浮点数:\n");
12
13    return 0;
14 }

```

```

选择Microsoft Visual Studio 调试控制台
float除以0: inf
float除以极小浮点数: inf
C:\Users\dell\Documents\C&C++\CSAPP\float0\Debug\float0.exe (进程 4452) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。

```

```

输出
显示输出来源(S): 调试
进程 (x2888) 已退出, 返回值为 0 (0x0)。
"float0.exe" (Win32): 已加载 "C:\Windows\System
"float0.exe" (Win32): 已加载 "C:\Windows\System32\user32.dll"。
"float0.exe" (Win32): 已加载 "C:\Windows\System32\user32.dll"。
进程 (x5314) 已退出, 返回值为 0 (0x0)。
进程 (x1544) 已退出, 返回值为 0 (0x0)。
程序 "[4452] float0.exe" 已退出, 返回值为 0 (0x0)。

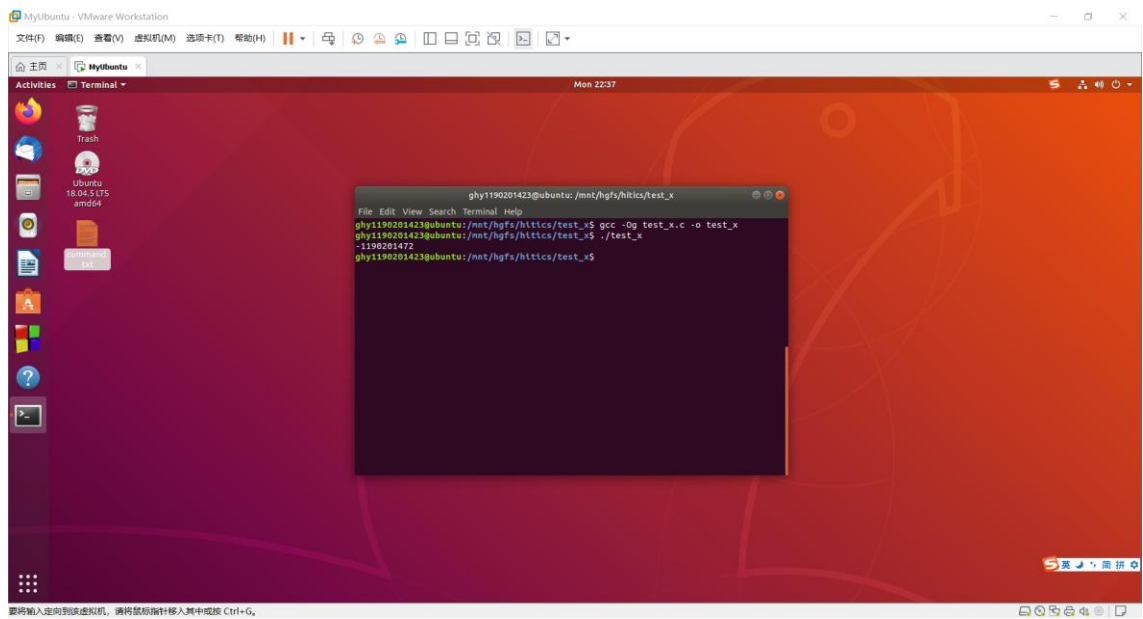
```

见附件 float0.c

7.4 类型转换的坑

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=-1190201423`, 现 `x=-1190201472`



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。

是哪些数据呢？绝对值小于等于 24 次方或者大于 2 的 24 次方但除最高的 24 位末尾都是 0。

```
ghy1190201423@ubuntu:/mnt/hgfs/hitacs/test_int_num$ vim int_float_num.c
ghy1190201423@ubuntu:/mnt/hgfs/hitacs/test_int_num$ gcc -Og int_float_num.c -o i
nt_float_num
ghy1190201423@ubuntu:/mnt/hgfs/hitacs/test_int_num$ ./int_float_num
150994944
ghy1190201423@ubuntu:/mnt/hgfs/hitacs/test_int_num$
```

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！

```

File Edit View Search Terminal Help
ghy1190201423@ubuntu: /mnt/hgfs/htlcs/test_int_num

#include <stdio.h>
#include <stdlib.h>
typedef unsigned char *byte_pointer;

void show_byte(byte_pointer start, size_t len)
{
    size_t i;
    for (size_t i = 0; i < len; i++)
    {
        printf("%.2x", start[i]);
    }
    printf("\n");
}

void show_float(float x)
{
    show_byte((byte_pointer)&x, sizeof(float));
}

int main()
{
    float a = 16777217; //1 0000 0000 0000 0000 0000 0001  若向偶数舍入, 应该显示为00 00 80 4b
    float b = 16777219; //1 0000 0000 0000 0000 0000 0011  若向偶数舍入, 应该显示为02 00 80 4b

    show_float(a);
    show_float(b);

    return 0;
}

```

22,104-99 All

```

ghy1190201423@ubuntu: /mnt/hgfs/htlcs/test_int_num$ vim test_float_even.c
ghy1190201423@ubuntu: /mnt/hgfs/htlcs/test_int_num$ ./test_float_even
0000804b
0200804b
ghy1190201423@ubuntu: /mnt/hgfs/htlcs/test_int_num$

```

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): $-(2^{-126}-2^{-149}) \sim (2^{-126}-2^{-149})$ 、 2^{-149} 、 $-(2^{127}-2^{104}) \sim -2^{127}$ 和 $2^{127} \sim (2^{127}-2^{104})$ 、 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 1.401298464e-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 2.028240960e31

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。

比较方法:

- 先定义一个很小的接近 0 的常量 (一般定义为 $1e-7$), 然后可以判断两个浮点数的差值是否小于这个常量。若小于, 则可以认为这两个浮点数相等。若大于, 则认为二者不相等。在对两个浮点数进行大小比较时, 由于浮点数并非真正意义上的实数, 只是其在某个范围内的近似, 因此在比较时, 不能简单地使用大于小于号进行比较, 应该判断连个浮点数差值的绝对值是否近似为 0。

- b) 当然，也可以按这个浮点数的二进制表示来进行比较：
1. 首先判断是否为 NAN，若为两个数都为 NAN， 则无法比较
 2. 再判断两个数符号位，如果一个为正数，一个负数，则正数大于负数，若为 0，则正 0 和 0 相等
 3. 然后取出阶码部分，若阶码为正数，则阶码大的数大，若为负数，则阶码小的数小
 4. 取出尾数比较，若为为负数，则尾数大的小，若为正数，尾数小的大。

第 8 章 舍尾平衡的讨论

8.1 描述可能出现的问题

舍位处理往往会采取四舍五入计算，这时就会产生误差，而如果报表中有这些数据的合计数值，那么舍位时产生的误差就会积累，有可能导致舍位过的数据与其合计值无法匹配。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。

举例说明：

$13,451.00 \text{ 元} + 45,150.00 \text{ 元} + 2,250.00 \text{ 元} - 5,649.00 \text{ 元} = 55,202.00 \text{ 元}$

现在单位变成单位万元，仍然保留两位小数，根据 4 舍 5 入的原则：

$1.35 \text{ 万元} + 4.52 \text{ 万元} + 0.23 \text{ 万元} - 0.56 \text{ 万元} = 5.54 \text{ 万元}$

出现 0.02 万的误差，平衡被打破。

8.2 给出完美的解决方案

我认为有一种解决该方法：因为作为税务局，要统计的数目是非常庞大的，所以我们下面的讨论建立在数据量很大的前提之下：

一般要完成的工作的数据的求和，如上文提到过的那样，舍尾可能会造成误差，因为税务局处理数据的时候单位一般都是万元，百万元甚至亿元，因此有一点小小的误差造成的影响也是非常巨大的，所以误差尽量越小越好，在统计数据时，由于人名币面额的固定性，所以‘元’之后最多有 2 位小数位，因此以‘万元’作为单位，最多会到小数点后 6 位，然后我认为数据是具有随机性的，因此不考虑一些很奇怪的小数点后组合，我认为可以有以下的方法：

先来定义一些符号：

堆	H
堆中元素个数	m
堆中元素	$H_i(i \geq 1 \& i \leq m)$
定义一个小常量(eg: $1e-6$)	E
要运算的数据	S_i

从第一个数据开始，定义一个 H 专门来存储小数点后的数字，先分配 H_1 存储小数点后的数字，当 H_1 数字达到与 1 之间的距离小于 E 的时候就可以进位为 1，如果 H_1 加上 S_i 后与 1 的距离大于 E，那么就可以分配 H_2 ，继续累积上面的操作，最后会得到一些值都很近似于 1 的 H，由于这些值都是随机的，设想 m 的个数不会太多，最后可以对 H 的每一个元素 H_i 进行穷举尝试，当其中 k 个元素的值与 k 的距离小于 k，就可以进行合并操作，最后剩下的几个数据结构值可以通过单边或者双边舍入平衡来进行求和。

第 9 章 总结

9.1 请总结本次实验的收获

本次实验我学习到了各种变量在内存中的地址，了解了各种变量在不同操作系统和不同位数机器中的大小，学会了怎么用反汇编的方式查看变量在内存中的地址和内容，明白了内存由数据段、代码段、堆栈段等段组成，知道了反汇编的操作步骤，知道了字符的 UTF-8 编码方式，学会了用 C 语言实现字符和浮点数、整数之间的转换，学会用 IEEE 754 编码构造浮点数的各种边界范围值，从而表达出浮点数的全部范围，理解了舍位平衡的概念。

9.2 请给出对本次实验内容的建议

希望实验的 ppt 的问题表述方式能够更改一下，很多句子模棱两可，容易无法理解或产生歧义。

希望能够减少重复实验内容，比如 3.2 和 3.3 中都要求打印 x, y, z 的地址和内容。

希望老师能提前说明一些代码的坑，utf8len.c 只能在 Linux 环境下运行，我在 VS 里运行还以为电脑坏了。

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>.