哈工大计算机系统实验二——DataLab数据表示

原创

dream or nightmare 于 2021-01-11 13:32:15 发布 10230 收藏 31
分类专栏： 哈工大计算机系统 文章标签： 哈工大计算机系统实验二 哈工大计算机系统实验2 哈工大计算机系统实验 csapp DataLab数据表示

> 计算机系统实验二特别难，和上一届的实验不一样，没有学长的火炬，当时做的时候特别崩溃。幸好有一帮志同道合的伙伴们，一起慢慢把实验解决了。
>
> 把火炬传下去！

实验报告

实 验（二）

题　　目 DataLab 数据表示

专　　业 计算机类

学　　号 xxxx

班　　级 xxxx

学　　生 xxxx

指 导 教 师 xxxx

实 验 地 点 xxxx

实 验 日 期 2019.9.25

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

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

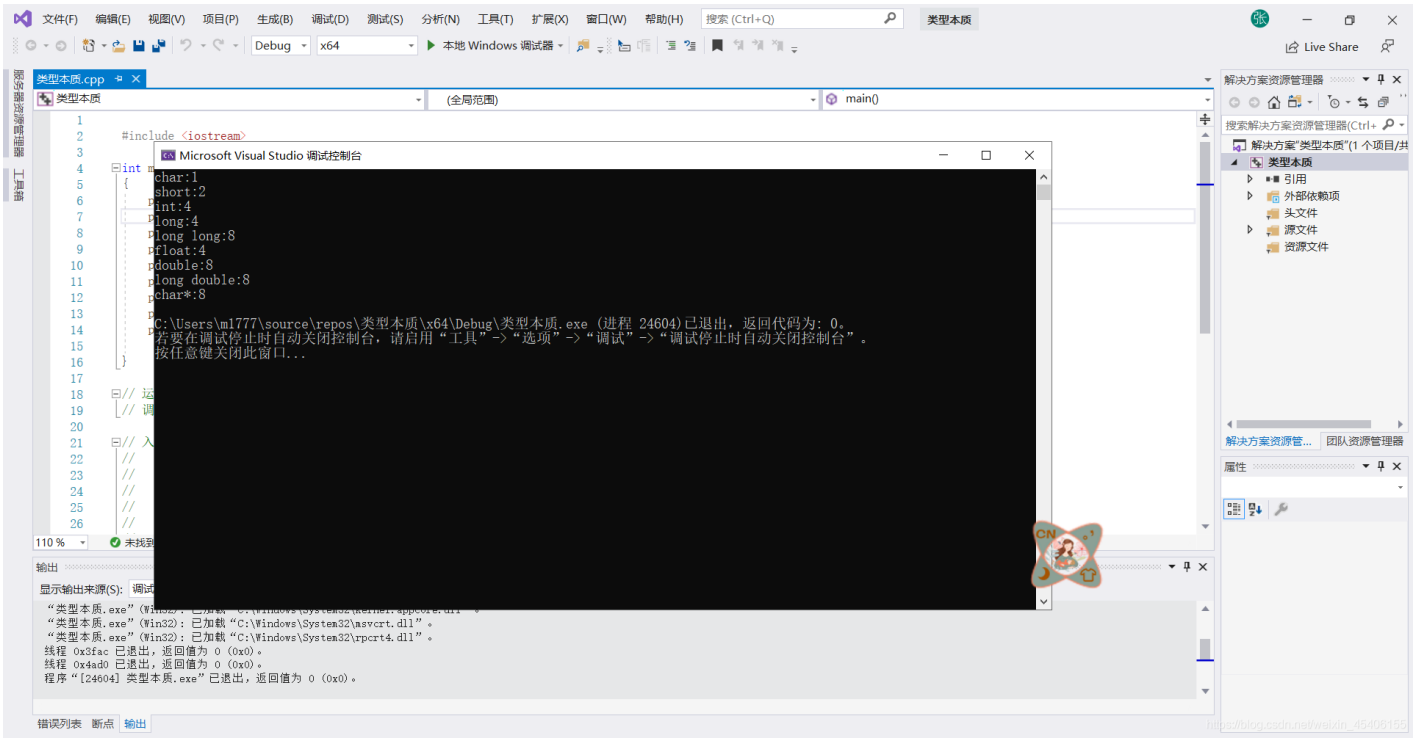
1.3 实验预习

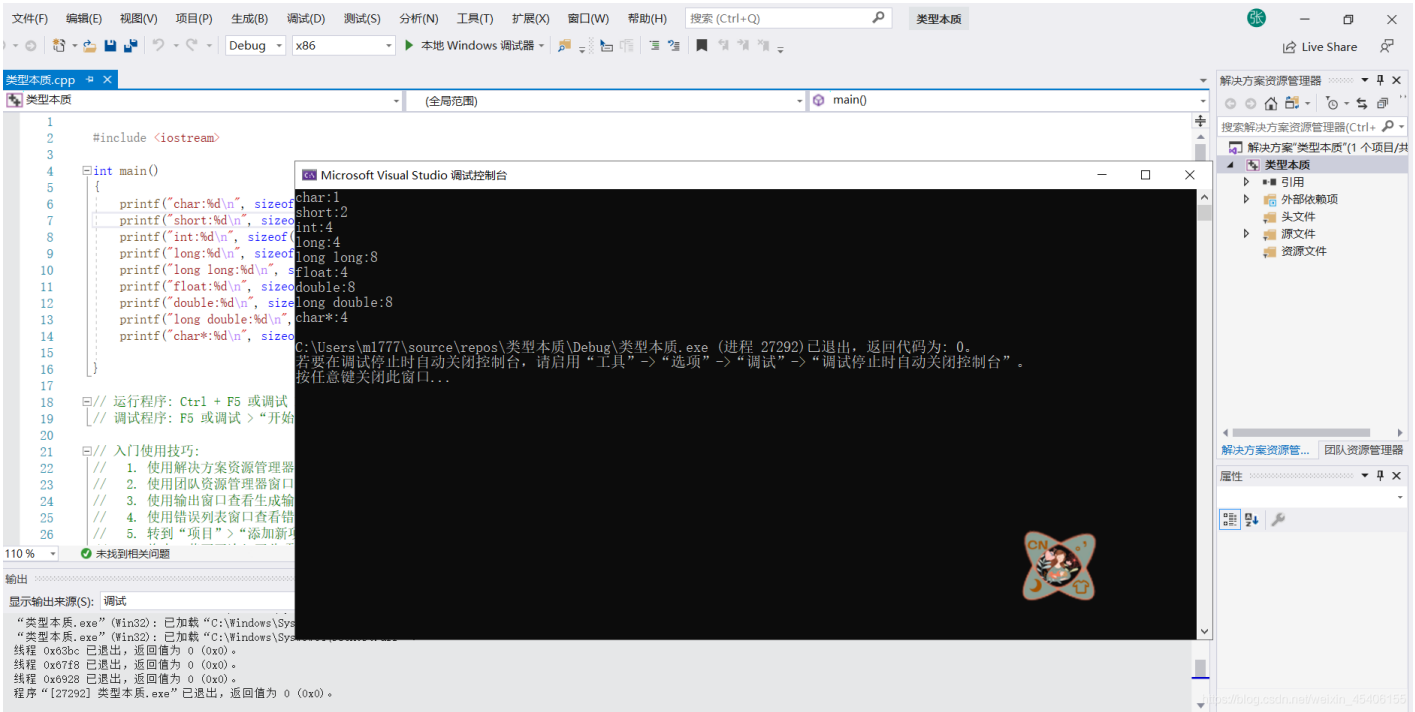
上实验课前，必须认真预习实验指导书（PPT或PDF）

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

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。

47 47 48 48

写出float/double类型最小的正数、最大的正数（非无穷）

0 11111110 11111111111111111111111: 表示float类型最大的正数,即2^(128)=3.4*10^(38)

0 00000001 00000000000000000000000: 表示float最小的正数,即2^(-126)=1.18*10^(-38)

double最大正数：1.79769*10^(308）

double最小正数：2.22507*10^(-308）

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（区域长度/表示的浮点个数）：_-(2-2^-23)*2^-126~(2-2^-23)*2^-126_____、_____2^-149_____、___-(2-2^-23)*2^127~2^127　　和　2^127~(2-2^-23)*2^127_____、___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

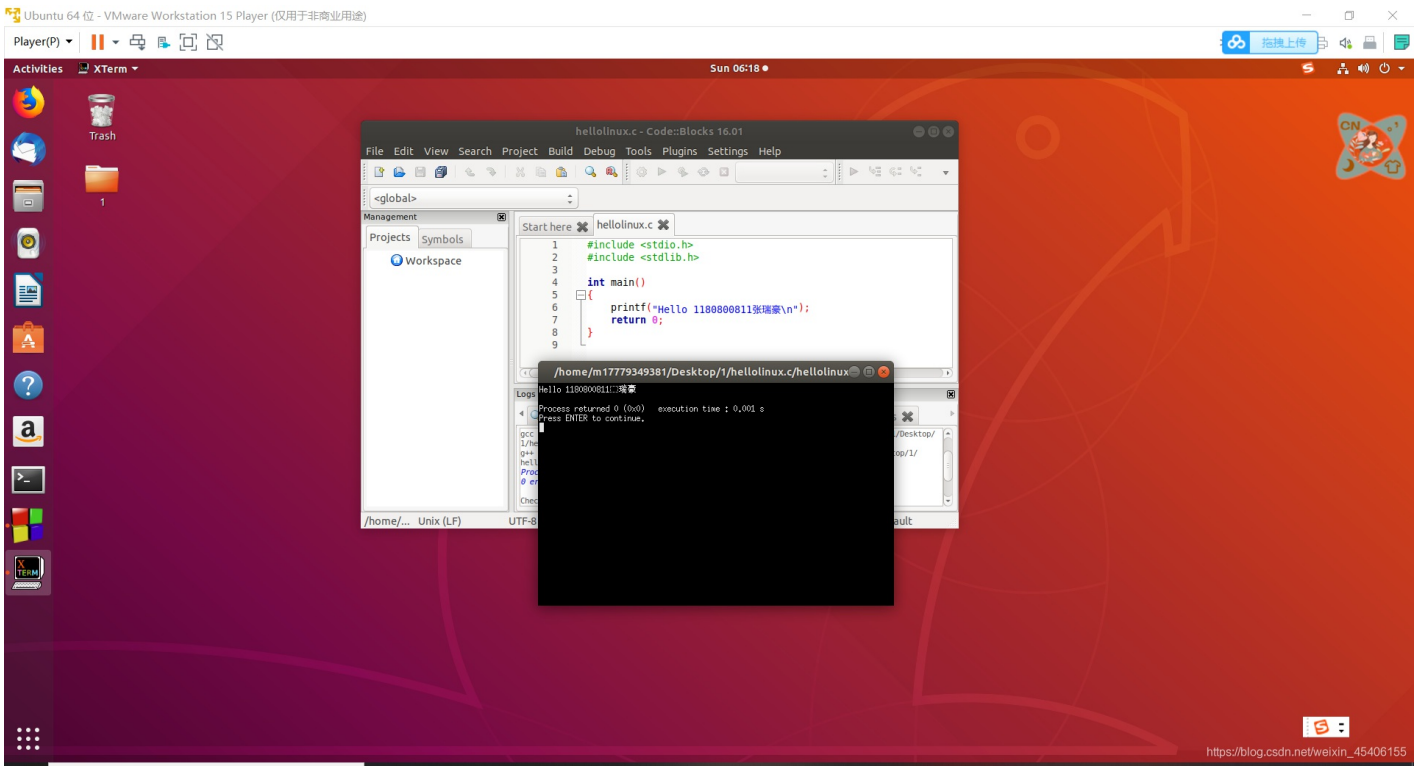


图2-1 Ubuntu下CodeBlocks截图

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。

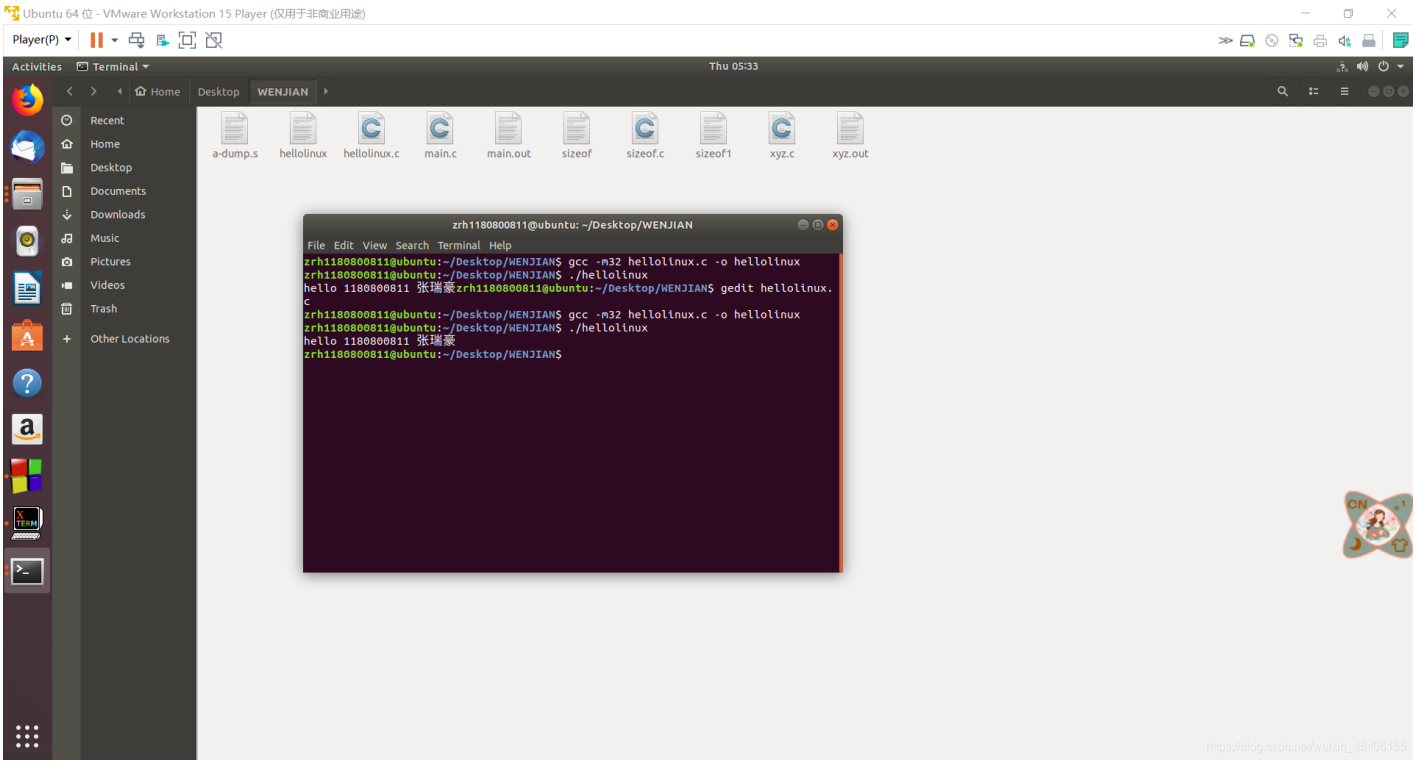


图2-2 Ubuntu与Windows共享目录截图

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/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

编译器对sizeof的实现方式：sizeof不是函数，也不是操作符。因为在编译运行时它没有被编译成机器指令，可以把它看做成是一个宏。

```
zrh1180800811@ubuntu:~/Desktop/WENJIAN$ ./sizeof
char:1
short:2
int:4
long:4
long long:8
float:4
double:8
long double:12
char*:4
```

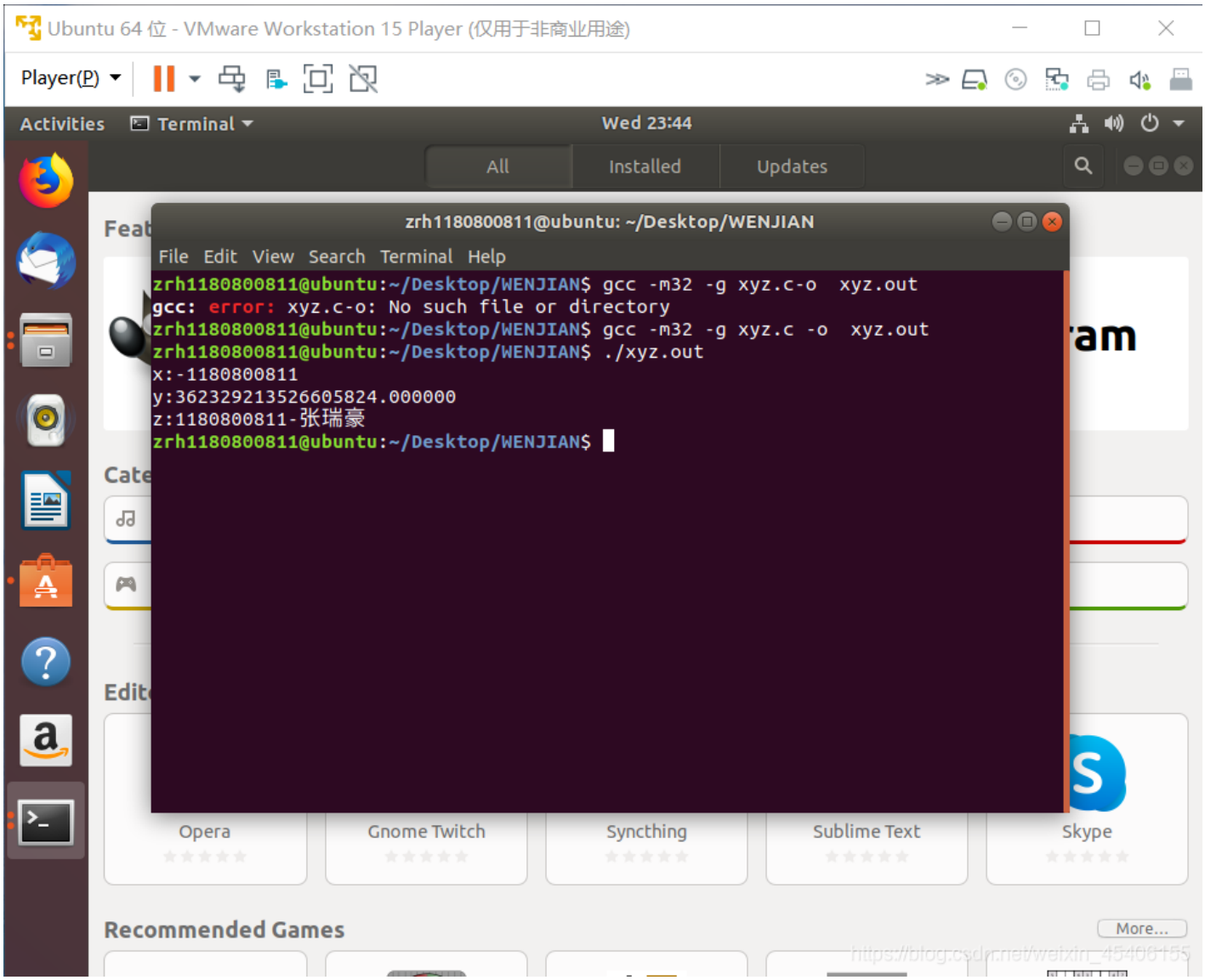
```
zrh1180800811@ubuntu:~/Desktop/WENJIAN$ ./sizeof1
char:1
short:2
int:4
long:8
long long:8
float:4
double:8
long double:16
char*:8
```

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




```
(gdb) p &x
$2 = (int *) 0x56557020 <x>
```

未设置断点前查看的全局变量x的地址如上

```
(gdb) x/4xb &x
0x56557020 <x>: 0xd5    0x68    0x9e    0xb9
(gdb) x/4xb &y
0xffffd06c:     0xc1    0x55    0x55    0x56
(gdb) x/4xb &z
0x56557040 <z.1868>:    0x31    0x31    0x38    0x30
(gdb) x/8xb &z
0x56557040 <z.1868>:    0x31    0x31    0x38    0x30    0x38    0x30    0x30    0
x38
```

设置断点进入main函数后查看的x、y、z地址如上

```
000000000000064a <main>:
 64a:   55                      push   %rbp
 64b:   48 89 e5                mov    %rsp,%rbp
 64e:   48 83 ec 10             sub    $0x10,%rsp
 652:   f3 0f 10 05 ee 00 00    movss  0xee(%rip),%xmm0        # 748 <_IO_stdin_used+0x18>
 659:   00
 65a:   f3 0f 11 45 fc          movss  %xmm0,-0x4(%rbp)
 65f:   8b 05 bb 09 20 00       mov    0x2009bb(%rip),%eax        # 201020 <x>
 665:   89 c6                   mov    %eax,%esi
 667:   48 8d 3d c6 00 00 00    lea    0xc6(%rip),%rdi        # 734 <_IO_stdin_used+0x4>
 66e:   b8 00 00 00 00          mov    $0x0,%eax
 673:   e8 a8 fe ff ff          callq  520 <printf@plt>
 678:   f3 0f 5a 45 fc          cvtss2sd -0x4(%rbp),%xmm0
 67d:   48 8d 3d b6 00 00 00    lea    0xb6(%rip),%rdi        # 73a <_IO_stdin_used+0xa>
 684:   b8 01 00 00 00          mov    $0x1,%eax
 689:   e8 92 fe ff ff          callq  520 <printf@plt>
 68e:   48 8d 35 ab 09 20 00    lea    0x2009ab(%rip),%rsi        # 201040 <z.2251>
 695:   48 8d 3d a4 00 00 00    lea    0xa4(%rip),%rdi        # 740 <_IO_stdin_used+0x10>
 69c:   b8 00 00 00 00          mov    $0x0,%eax
 6a1:   e8 7a fe ff ff          callq  520 <printf@plt>
 6a6:   b8 00 00 00 00          mov    $0x0,%eax
 6ab:   c9                      leaveq
 6ac:   c3                      retq
 6ad:   0f 1f 00                nopl   (%rax)
```


64位下的反汇编代码表示如上图

```
0000051d <main>:
 51d:   8d 4c 24 04             lea    0x4(%esp),%ecx
 521:   83 e4 f0                and    $0xfffffff0,%esp
 524:   ff 71 fc                pushl  -0x4(%ecx)
 527:   55                      push   %ebp
 528:   89 e5                   mov    %esp,%ebp
 52a:   53                      push   %ebx
 52b:   51                      push   %ecx
 52c:   83 ec 10                sub    $0x10,%esp
 52f:   e8 ec fe ff ff          call   420 <__x86.get_pc_thunk.bx>
 534:   81 c3 a4 1a 00 00       add    $0x1aa4,%ebx
 53a:   d9 83 5c e6 ff ff       flds   -0x19a4(%ebx)
 540:   d9 5d f4                fstps  -0xc(%ebp)
 543:   8b 83 48 00 00 00       mov    0x48(%ebx),%eax
 549:   83 ec 08                sub    $0x8,%esp
 54c:   50                      push   %eax
 54d:   8d 83 48 e6 ff ff       lea    -0x19b8(%ebx),%eax
 553:   50                      push   %eax
 554:   e8 57 fe ff ff          call   3b0 <printf@plt>
 559:   83 c4 10                add    $0x10,%esp
 55c:   d9 45 f4                flds   -0xc(%ebp)
 55f:   83 ec 04                sub    $0x4,%esp
 562:   8d 64 24 f8             lea    -0x8(%esp),%esp
 566:   dd 1c 24                fstpl  (%esp)
 569:   8d 83 4e e6 ff ff       lea    -0x19b2(%ebx),%eax
 56f:   50                      push   %eax
 570:   e8 3b fe ff ff          call   3b0 <printf@plt>
 575:   83 c4 10                add    $0x10,%esp
 578:   83 ec 08                sub    $0x8,%esp
 57b:   8d 83 68 00 00 00       lea    0x68(%ebx),%eax
 581:   50                      push   %eax
 582:   8d 83 54 e6 ff ff       lea    -0x19ac(%ebx),%eax
 588:   50                      push   %eax
 589:   e8 22 fe ff ff          call   3b0 <printf@plt>
 58e:   83 c4 10                add    $0x10,%esp
 591:   b8 00 00 00 00          mov    $0x0,%eax
 596:   8d 65 f8                lea    -0x8(%ebp),%esp
 599:   59                      pop    %ecx
 59a:   5b                      pop    %ebx
 59b:   5d                      pop    %ebp
 59c:   8d 61 fc                lea    -0x4(%ecx),%esp
 59f:   c3                      ret
```

32位下的反汇编代码如上

如图，x和z在代码段使用的时候，是通过ebx加上一个偏移地址来从相应的内存区出来的，而ebx指向了数据段，而y在代码段调用的时候，发现通过ebp，也就是栈指针加偏移地址从栈取出来的。

x与y在___汇编_____阶段转换成补码与ieee754编码。

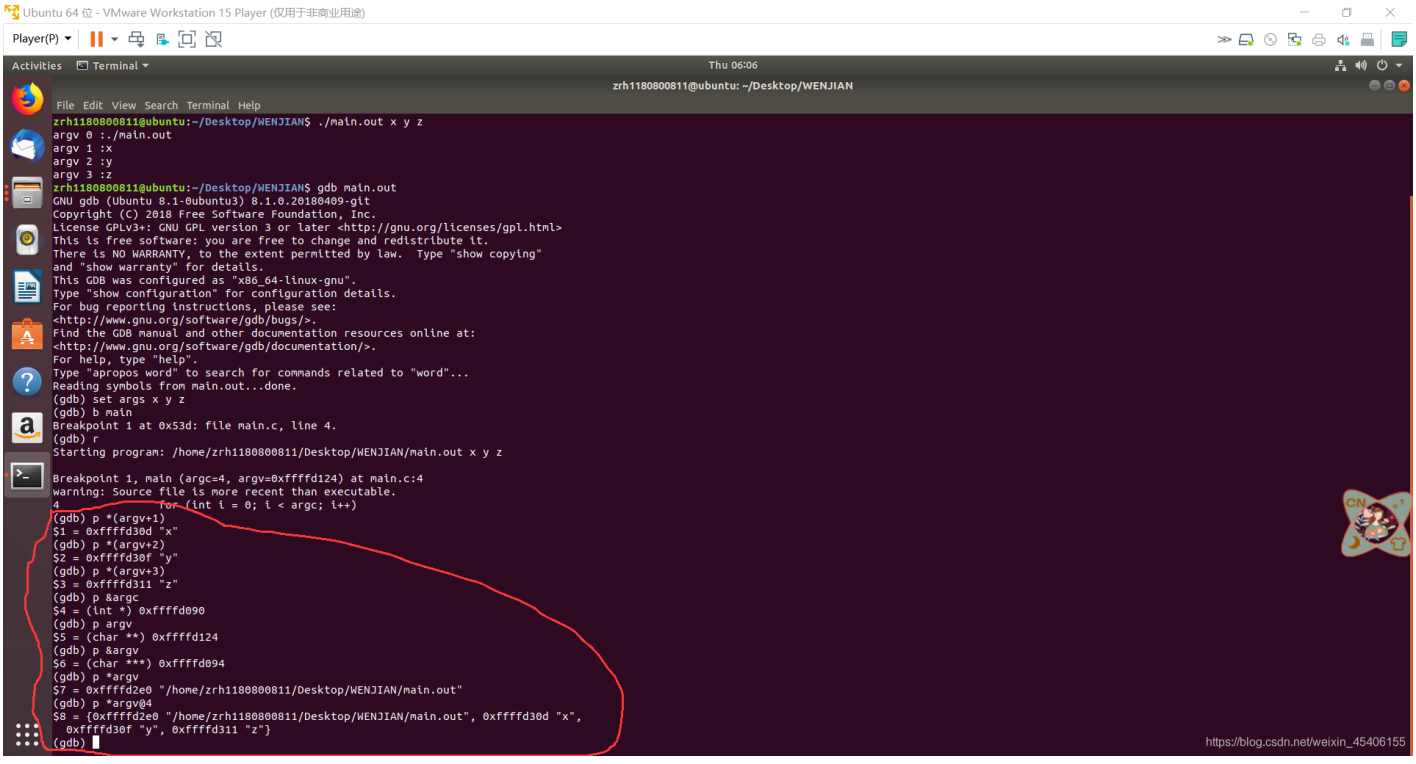
数值型常量与变量在存储空间上的区别是：__宏常量不进行存储，const常量存储在代码段，静态变量和全局变量存储在数据段，局部变量存储在堆栈段_____

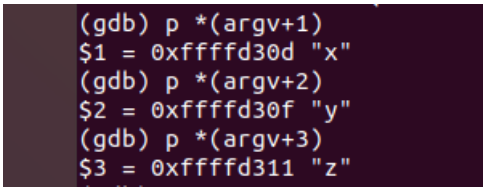
字符串常量与变量在存储空间上的区别是：__字符串常量保存在代码段，静态变量和全局变量保存在数据段，局部变量在堆栈段_____

常量表达式在计算机中处理方法是：_____在编译阶段转换成常量来存储_____

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的地址如图

```
(gdb) p &argc
$4 = (int *) 0xffffd090
```

argc的地址如图

```
(gdb) p argv
$5 = (char **) 0xffffd124
(gdb) p &argv
$6 = (char ***) 0xffffd094
```

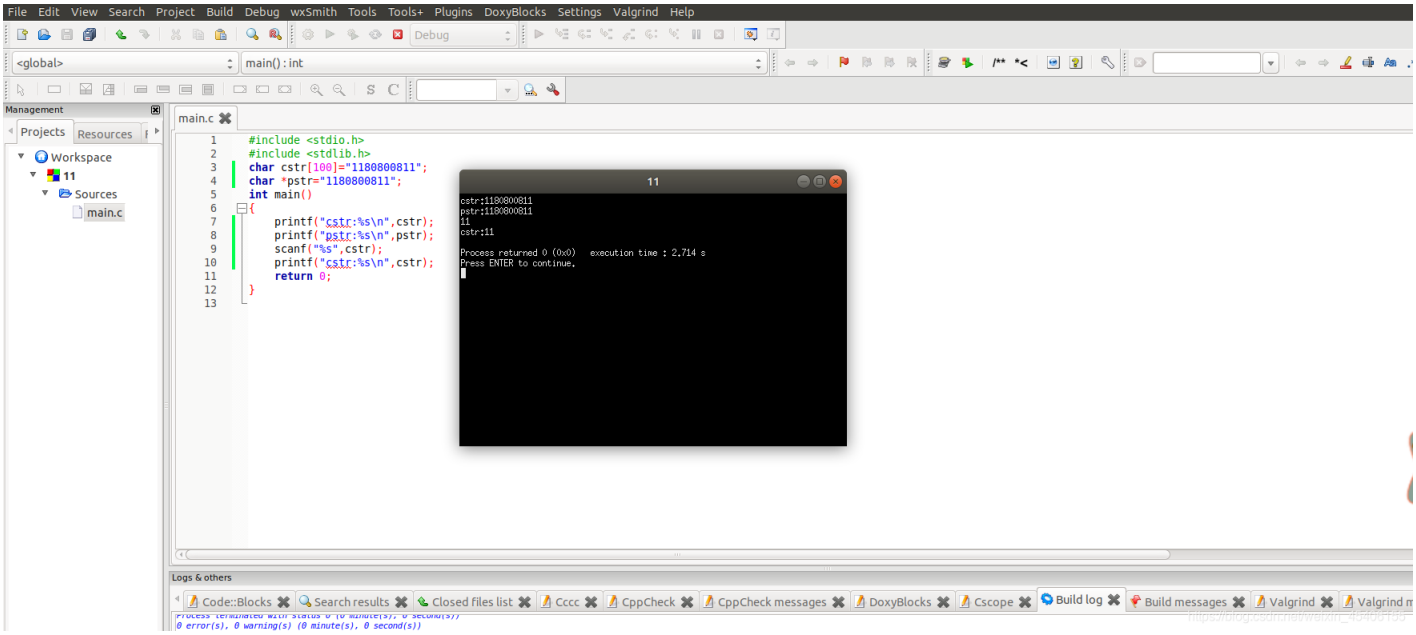
argv的地址和内容如图

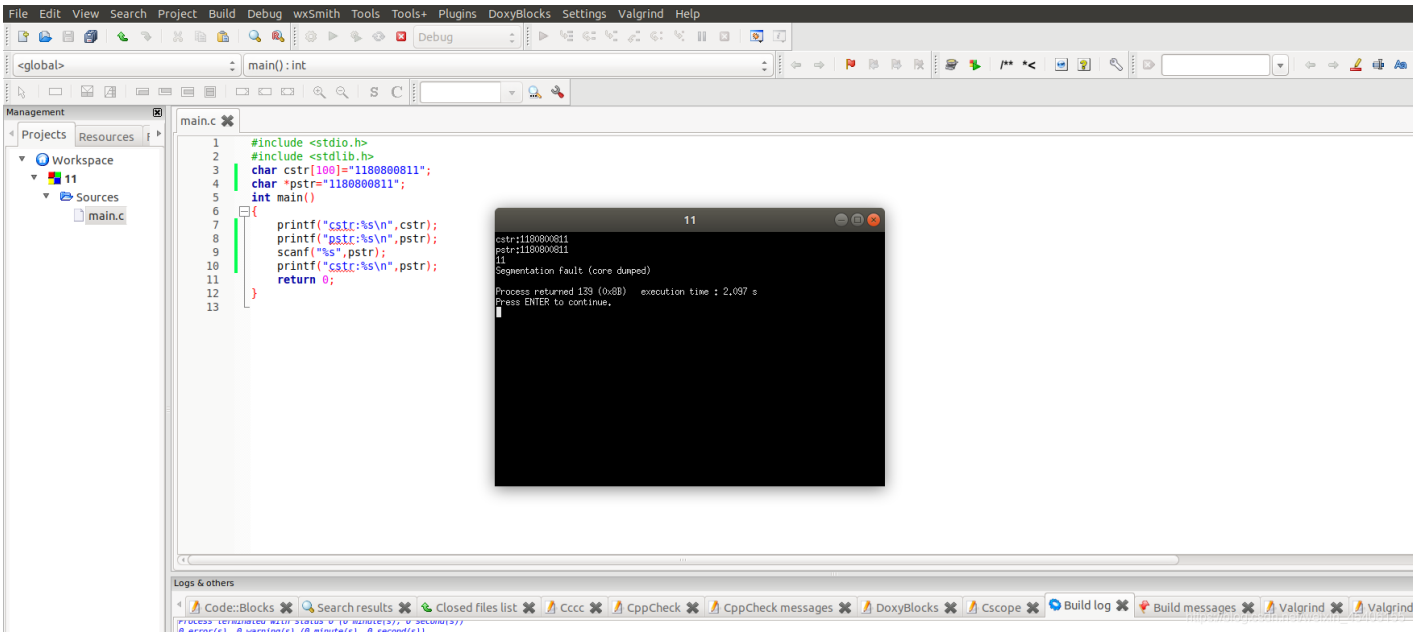
3.4 指针与字符串的区别

cstr的地址与内容截图，pstr的内容与截图，截图5

这个就和下面出现的问题类似了，就是字符串指针赋不上值。

pstr修改内容会出现什么问题_____ 字符串指针为常量不可修改，修改会导致程序出错，无法运行,如图所示_____





第4章 深入分析UTF-8编码

4.1 提交utf8len.c子程序

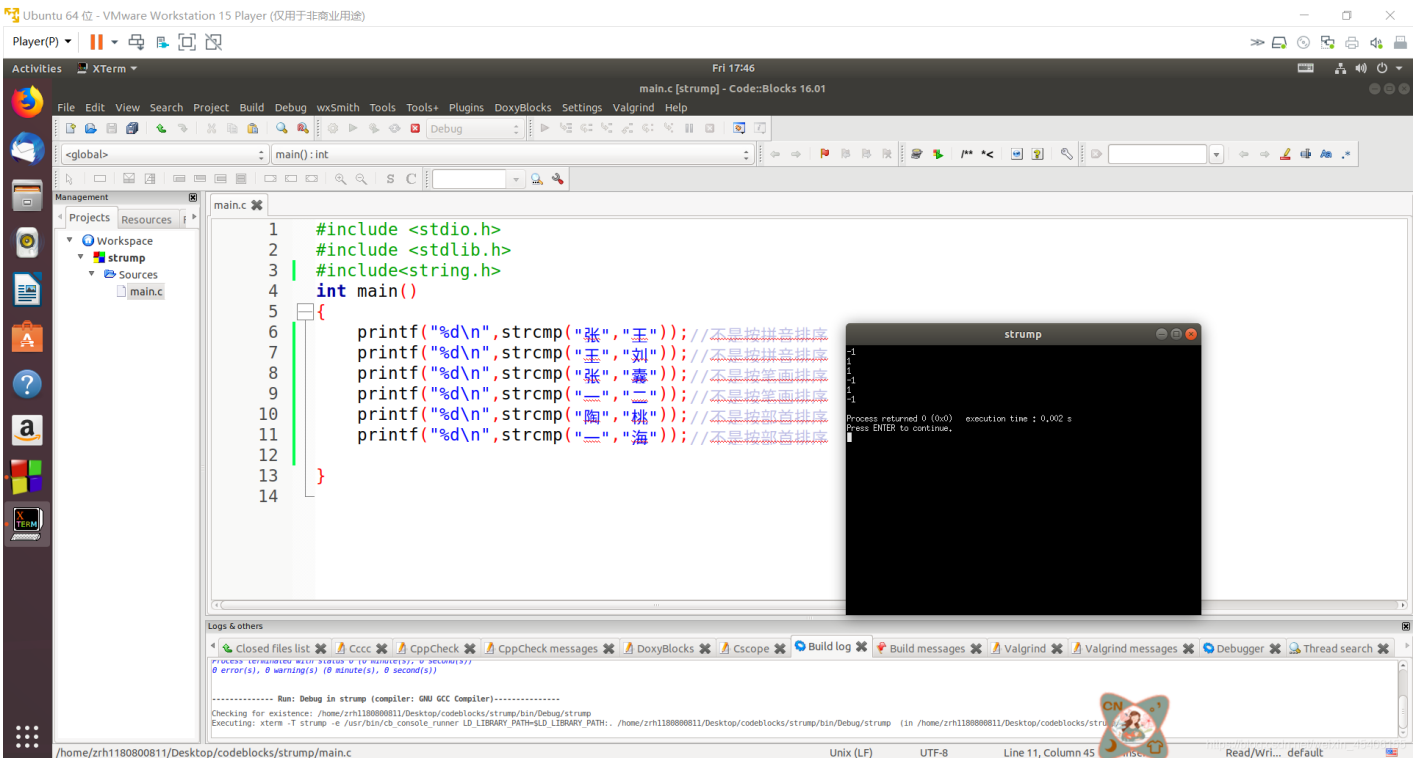
得在Linux系统才能运行,在Windows下CB和vs均会报错。（gets）

```
// utf8len.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int utf8len(char* a);
int main()
{
 int word = 0;
 char str[1000], * p;
 printf("输入一个字符串：");
 gets(str);5
 p = str;
 word = utf8len(p);
 printf("\t%d", word);
 return 0;
}
int utf8len(char* a)
{
 int word = 0;
 int len = 0;
 for (int i = 0; a[i] != '\0'; i += len) {
  unsigned char byte = a[i];
  if (byte >= 0xFC)
   len = 6;
  else if (byte >= 0xF8)
   len = 5;
  else if (byte >= 0xF0)
   len = 4;
  else if (byte >= 0xE0)
   len = 3;
  else if (byte >= 0xC0)
   len = 2;
  else
   len = 1;
  word++;
 }
 return word;
}
```

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

按照康熙字典排序



如图，第一行printf比较“张”，“王”，strcmp值为-1，说明“张”排在“王”前面，第二行printf比较“王”，“刘”，strcmp值为1，说明“王”排在“刘”后面，如果按部首排序，则出现z排在w之前，而w排在l之后，显然不是按拼音顺序排序。

同理第三四行printf排除了按笔画排序的可能。

第五六行printf排除了按部首排序的可能。

经检验，上述的举例均符合康熙字典排序，故推测Linux系统下strcmp函数按照康熙字典的顺序对汉字进行排序。

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？

先看下面的笔画排序规则：

1. 首先按照笔画数由少到多排序的原则。姓氏笔画少的排在前面，姓氏笔画多的排在后面。比如王和李，王四画，李七画，则王在前李在后。
2. 笔画数相同的，按姓氏起次笔排序的的原则。按“一（横）”、“|（竖）”、“丿（撇）”、“捺”、“折”的顺序排序。比如干字第一笔为一，则在三画[一]类。“莫”字第一笔为一，第二笔为|，则在事画[一|]类。
3. 同姓一般以姓名的第二个字的笔画为多少序。例如，王大宁和王胜利，“大”的笔画少，排在前，“胜”的笔画多，排在后面。如果姓名是两个字，第二个字的笔画按零画对待。例如王绳和王大宁，王绳是两个字，而王大宁是三个字，则王绳排在前，王大宁排在后。复姓按单姓对待。两个名的第一个字笔画数相同，则看两个名的第二个字的笔画是多少来比较。
4. 姓氏的笔画数相同、起次笔顺序一致的，按姓氏的字形结构排序的原则。先左右形字，再上下形字，后整体形字。如：同时八画[| 折]，“明”在前，“昌”次之。国在后。
5. 对于姓氏的笔画数相同、起次笔顺序一致，且字形结构相同的，左右形汉字的排序要遵循——按“左偏旁”笔画数由少到多的顺序排定之原则；例如：7画“丽贡志芙吾”的排列。杂合形汉字的排序要遵循——按“杂合偏旁”重心所在点按逆时针顺序先后排定之原则，例如：9画“风逃勉”的排列。

然后可以这样来实现：

将汉字的Unicode编码和上述排序一一对应，通过读取其Unicode编码找到其对应的序号，进而进行排序。

第5章 数据变换与输入输出

5.1 提交cs_atoi.c

```
#include <stdio.h>
#include <stdlib.h>
#include<assert.h>
int my_strlen(char* str)

{

 assert(str != NULL);

 int count = 0;

 while (*str != '\0')

 {

  count++;

  str++;

 }

 return count;

}
int main()
{
 char a[80];
 scanf("%s", a);
 int b;
 int count = 1;
 int n = my_strlen(a);
 int sum = 0;
 for (int i = n - 1; i >= 0; i--) {
  b = (int)a[i];
  if (b >= 48 && b <= 57)
  {
   sum += count * (b - 48);
   count *= 10;
  }

 }
 printf("%d", sum);

}
```

5.2 提交cs_atof.c

```
    #include <stdio.h>
#include <stdlib.h>
#include<assert.h>
int my_strlen(char* str)

{
 assert(str != NULL);

 int count = 0;

 while (*str != '\0')

 {
  count++;
  str++;
 }
 return count;

}
int main()
{
 char str[100];
 scanf("%s", str);
 float sum = 0;
 float count = 1.0;
 int n = my_strlen(str);
 int x = 0;
 int i = 0;
 for ( i = 0; i < n&&str[i]!='.'; i++) {
  x++;
 }
 for ( i = x - 1; i >= 0; i--) {

  if ((int)str[i] >= 48 && (int)str[i]<= 57)
  {
   sum += count * ((int)str[i] - 48);
   count *= 10;
  }
 }
 count = 0.1;
 for ( i = x + 1; i < n; i++) {

  if ((int)str[i] >= 48 && (int)str[i]<= 57)
  {
   sum += count * ((int)str[i] - 48);
   count /= 10;
  }
 }
 printf("%lf",sum);
}
```

5.3 提交cs_itoa.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int x;
 scanf("%d", &x);
 int y=x;
  int n = 1;
  while(y/10)
 {
  y/=10;
  ++n;
 }

 char str[n];
 for (int i = n - 1; i >= 0; i--) {
  int m=x%10;
  x/=10;
  str[i] = m+48;
 }
 printf("%s", str);
}
```

5.4 提交cs_ftoa.c

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

const double eps = 1e-11;

void cs_ftoa(char *str,double num)
{
    int high;//float_整数部分
    double low;//float_小数部分
    char *start=str;
    int n=0;
    char ch[20];
    int i;
    high=(int)num;
    low=num-high;

    while(high>0){
        ch[n++]='0'+high%10;
        high=high/10;
    }

    for(i=n-1;i>=0;i--){
        *str++=ch[i];
    }

    num -= (int)num;
   double tp = 0.1;
    *str++='.';

    while(num > eps){//精度限制
        num -= tp * (int)(low * 10);
        tp /= 10;
        *str++='0'+(int)(low*10);
        low=low*10.0-(int)(low*10);
    }
    *str='\0';
    str=start;
}
int main()
{
    double a;
    printf("请输入一个浮点数：");
    while( ~scanf("%lf", &a) ) {
        char str[20];
       cs_ftoa(str,a);
        printf("%s\n\n",str);
    }

}
```

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：

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

#include<unistd.h>

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,void *buf,size_t n)

返回：若成功则为读的字节数，若出错则为-1

read函数从描述符为fd的当前文件位置复制最多n个字节到内存位置buf，返回值为-1表示一个错误，返回值为0表示EOF，否则，返回值为实际传送的字节数量

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。

[提示一下，这一部分得最后一章I/O系统才会讲到，老师超纲了！]

第6章 整数表示与运算

6.1 提交fib_dg.c

```
#include <stdio.h>
#include <stdlib.h>
    int Fib(int n)
{   if(n==0)   return 0;
 else if (n == 1 || n == 2)
 {
  return 1;
 }
 else
 {
  return Fib(n - 1) + Fib(n - 2);
 }
}
int main()
{
  int n=0;
  int sum=0;
  scanf("%d", &n);
  sum = Fib(n);
  printf("%d", sum);
  return 0;
}
```

6.2 提交fib_loop.c

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

unsigned int Fib(unsigned int n)
 {
     if (n == 0)  return 0;
  if (n == 2||n==1)  return 1;
  unsigned int f1 = 1; unsigned int f2 = 1; unsigned int c = 0;
  for (unsigned int i = 3; i <= n; i++)
  { c = f1 + f2;
   f1 = f2;
   f2 = c;
}
  return c;
 }
int main()
{
 unsigned int n = 0;
 unsigned int sum = 0;
 scanf_s("%d", &n);
 sum = Fib(n);
 printf("%d", sum);
 return 0;

}
```

6.3 fib溢出验证

int 时从n=____47_____时溢出，long时n=___47_____时溢出。

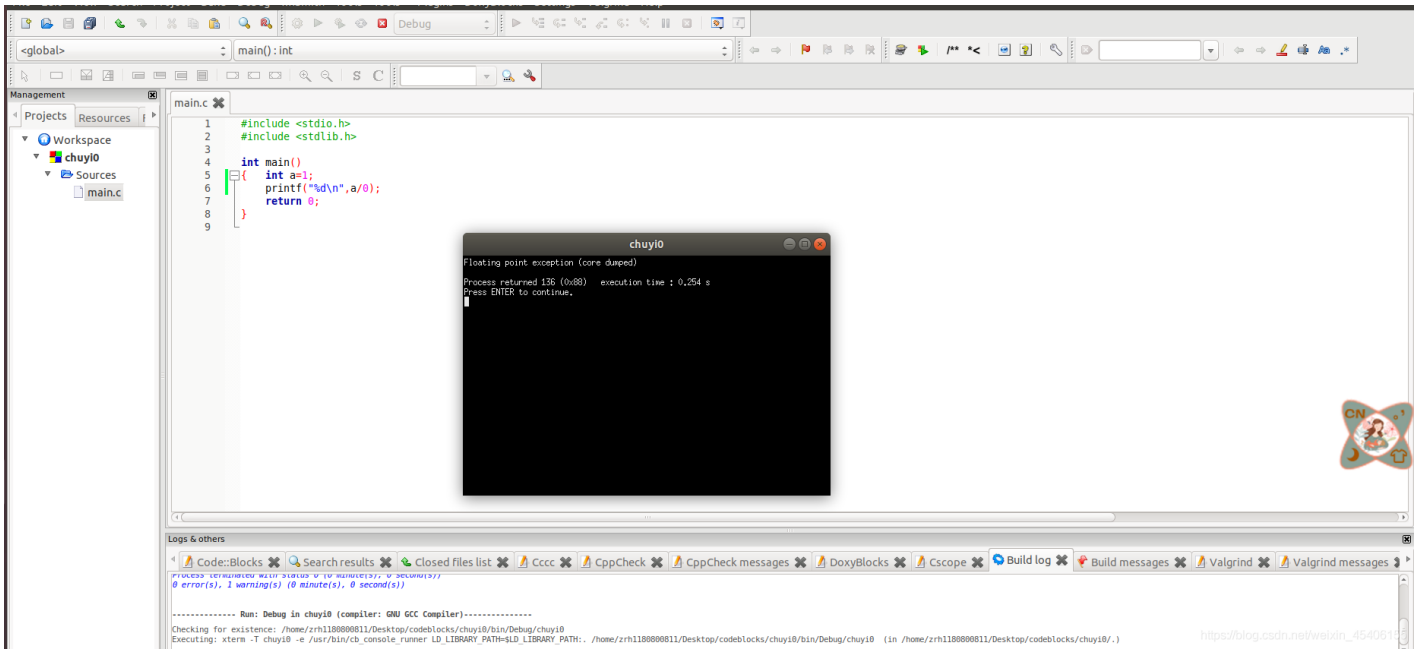
unsigned int 时从n=___48_____时溢出，unsigned long时n=__48_____时溢出。

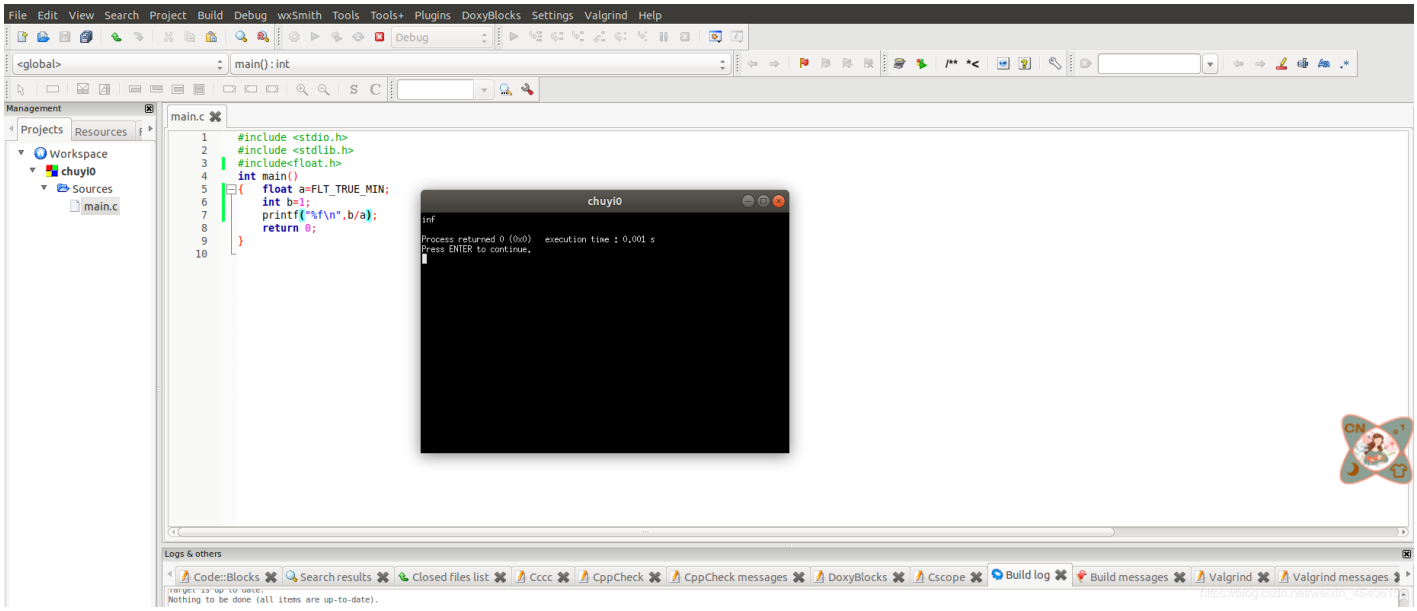
6.4 除以0验证：

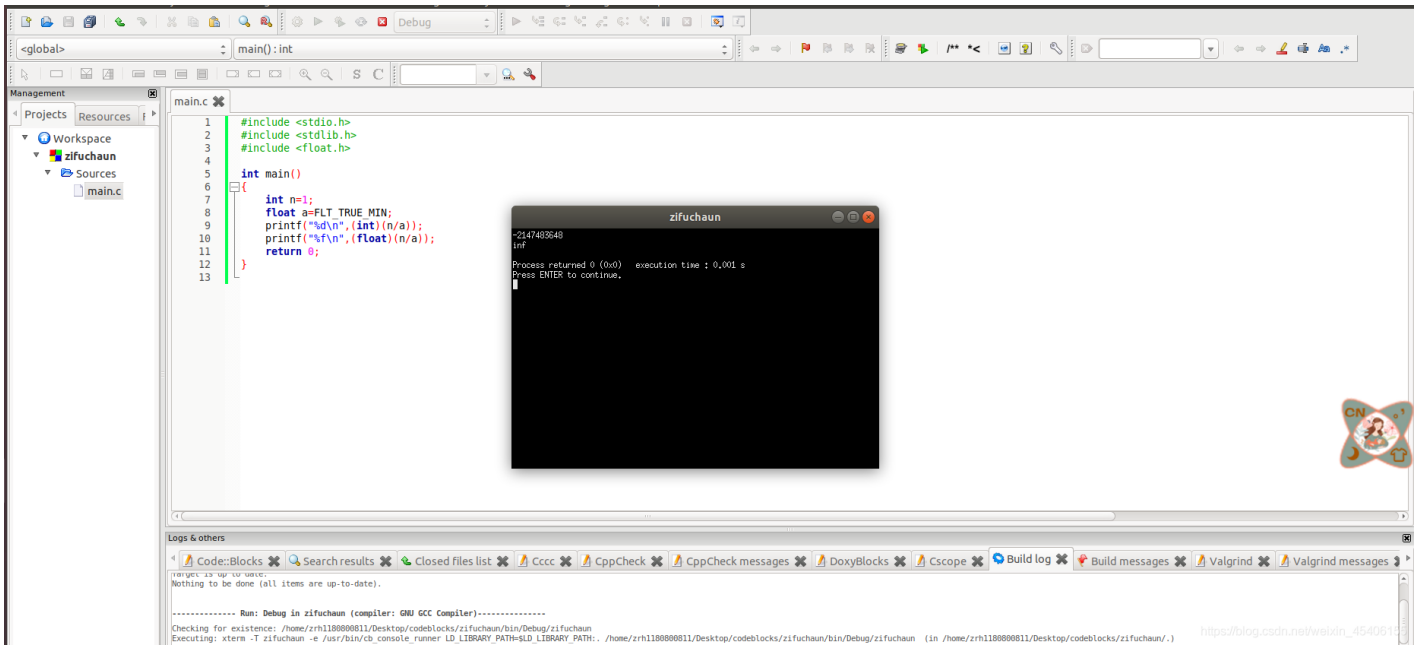
除以0：截图1

除以极小浮点数，截图：

除零程序终止，除极小浮点数以整型输出为最小int值，以浮点输出则为正无穷。


Management
Projects
Resources
Workspace
chuyi0
Sources
main.c
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6 int a=1;
7 printf("%d\n",a/0);
8 return 0;
9 }
chuyi0
Floating point exception (core dumped)
Process returned 136 (0x88) execution time : 0.254 s
Press ENTER to continue.
Logs & others
Code::Blocks
Search results
Closed files list
Cccc
CppCheck
CppCheck messages
DoxyBlocks
Cscope
Build log
Build messages
Valgrind
Valgrind messages
0 error(s), 1 warning(s) (0 minute(s), 0 second(s))
---------- Run: Debug in chuyi0 (compiler: GNU GCC Compiler)----------
Checking for existence: /home/zrh1180800811/Desktop/codeblocks/chuyi0/bin/Debug/chuyi0
Executing: xterm -T chuyi0 -e /usr/bin/cb_console_runner LD_LIBRARY_PATH=$LD_LIBRARY_PATH:. /home/zrh1180800811/Desktop/codeblocks/chuyi0/bin/Debug/chuyi0 (in /home/zrh1180800811/Desktop/codeblocks/chuyi0/.)


File Edit View Search Project Build Debug wxSmith Tools Tools+ Plugins DoxyBlocks Settings Valgrind Help
Management
Projects
Resources
Workspace
chuyi0
Sources
main.c
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include<float.h>
4 int main()
5 {
6 float a=FLT_TRUE_MIN;
7 int b=1;
8 printf("%f\n",b/a);
9 return 0;
10 }
chuyi0
inf
Process returned 0 (0x0) execution time : 0.001 s
Press ENTER to continue.
Logs & others
Code::Blocks
Search results
Closed files list
Cccc
CppCheck
CppCheck messages
DoxyBlocks
Cscope
Build log
Build messages
Valgrind
Valgrind messages
Nothing to be done (all items are up-to-date).


Management
Projects
Resources
Workspace
zifuchaun
Sources
main.c
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <float.h>
4
5 int main()
6 {
7 int n=1;
8 float a=FLT_TRUE_MIN;
9 printf("%d\n",(int)(n/a));
10 printf("%f\n",(float)(n/a));
11 return 0;
12 }
13
zifuchaun
-2147483648
inf
Process returned 0 (0x0) execution time : 0.001 s
Press ENTER to continue.
Logs & others
Code::Blocks
Search results
Closed files list
Cccc
CppCheck
CppCheck messages
DoxyBlocks
Cscope
Build log
Build messages
Valgrind
Valgrind messages
Nothing to be done (all items are up-to-date).
---------- Run: Debug in zifuchaun (compiler: GNU GCC Compiler)----------
Checking for existence: /home/zrh1180800811/Desktop/codeblocks/zifuchaun/bin/Debug/zifuchaun
Executing: xterm -T zifuchaun -e /usr/bin/cb_console_runner LD_LIBRARY_PATH=$LD_LIBRARY_PATH:. /home/zrh1180800811/Desktop/codeblocks/zifuchaun/bin/Debug/zifuchaun (in /home/zrh1180800811/Desktop/codeblocks/zifuchaun/.)

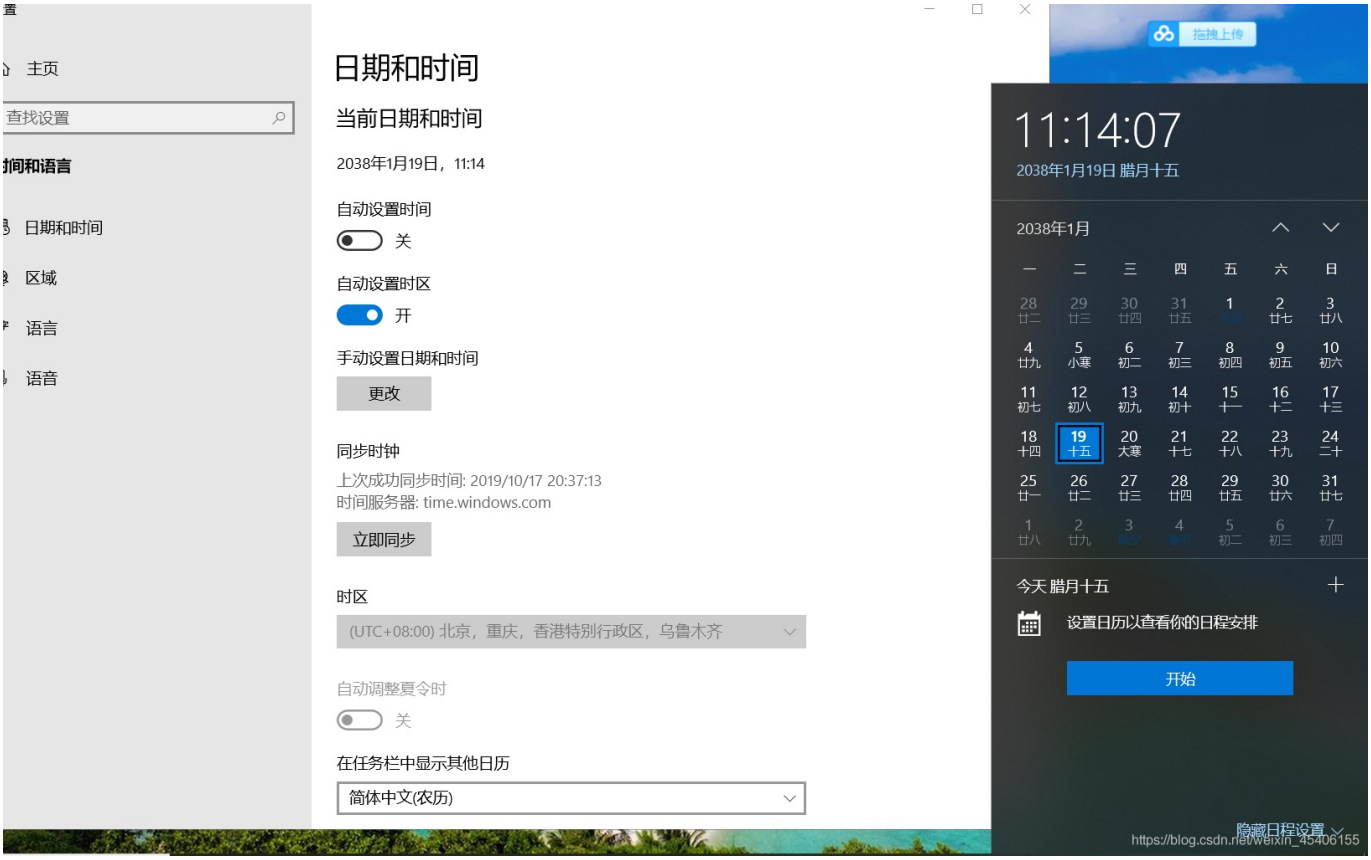
6.5 万年虫验证

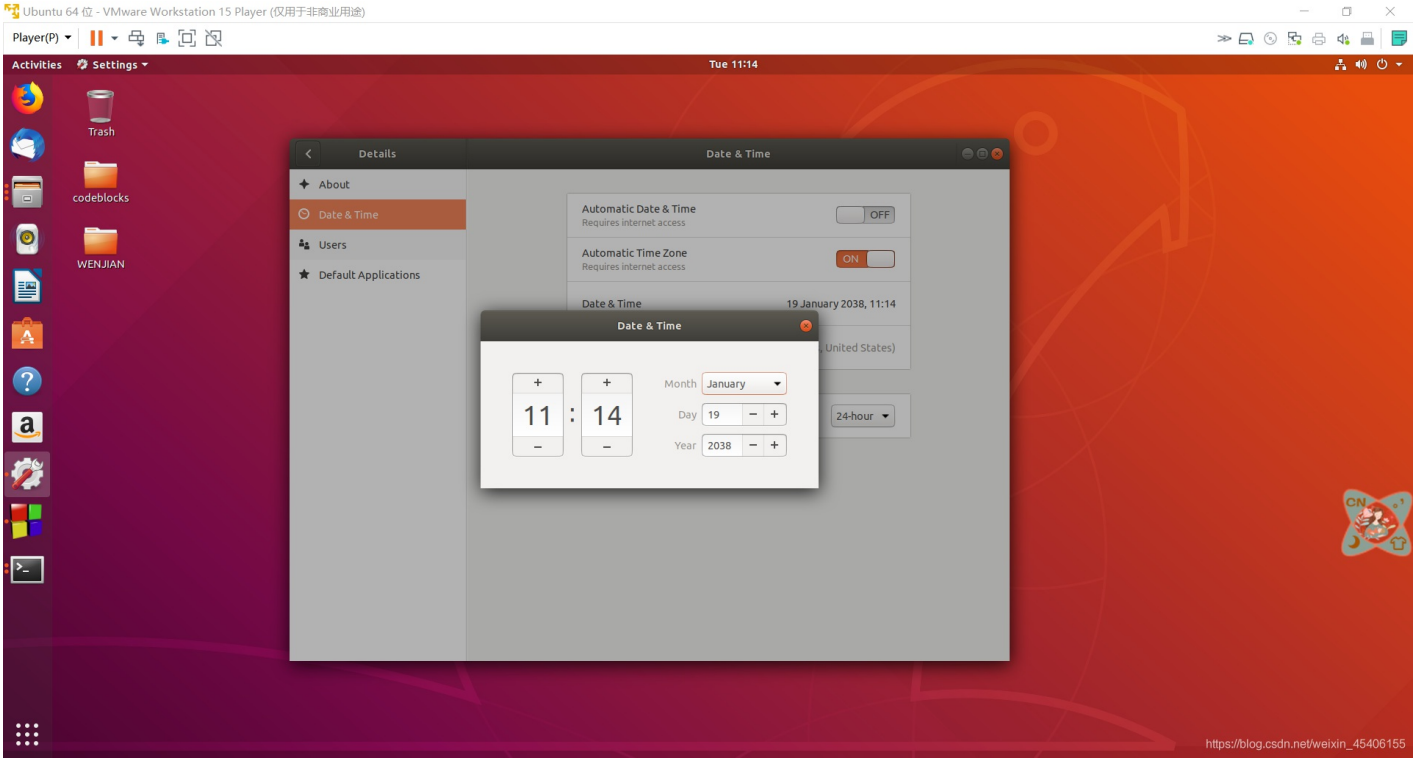
你的机器到9999年12月31日23:59:59后，时钟怎么显示的？Windows/Linux下分别截图：

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

2038直接改就行，64位系统下都没现象。





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。

-10.1

符号位：1

（1）整数二进制：1010

（2）小数部分二进制：0.00011001100110011001100110011

（3）科学记数法为：-1.0100 001 1001 1001 1001 1001*2^3

（4）指数：3，3+127=130，因此阶码为1000 0100

（5）小数部分采用乘2取整法得：

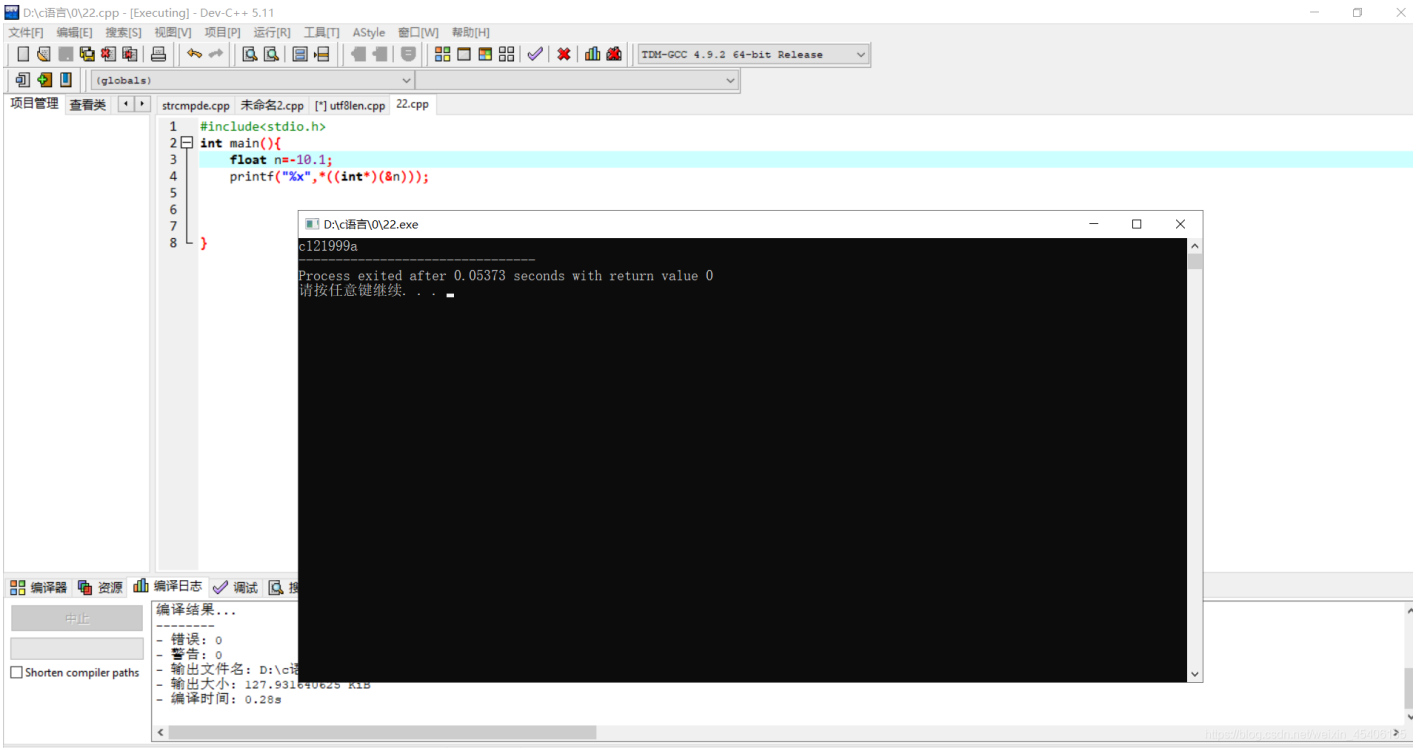
因此： 符号位　　指数部分　　尾数部分

1　　1000 0100　　0100001 1001 1001 1001 1001

因此从低到高的二进制为：1100 0010 0010 0001 1001 1001 1001 1001

对应的16进制为 ：c121999a

因此低到高字节排序为：9a 99 21 c1



7.2特殊float数据的处理

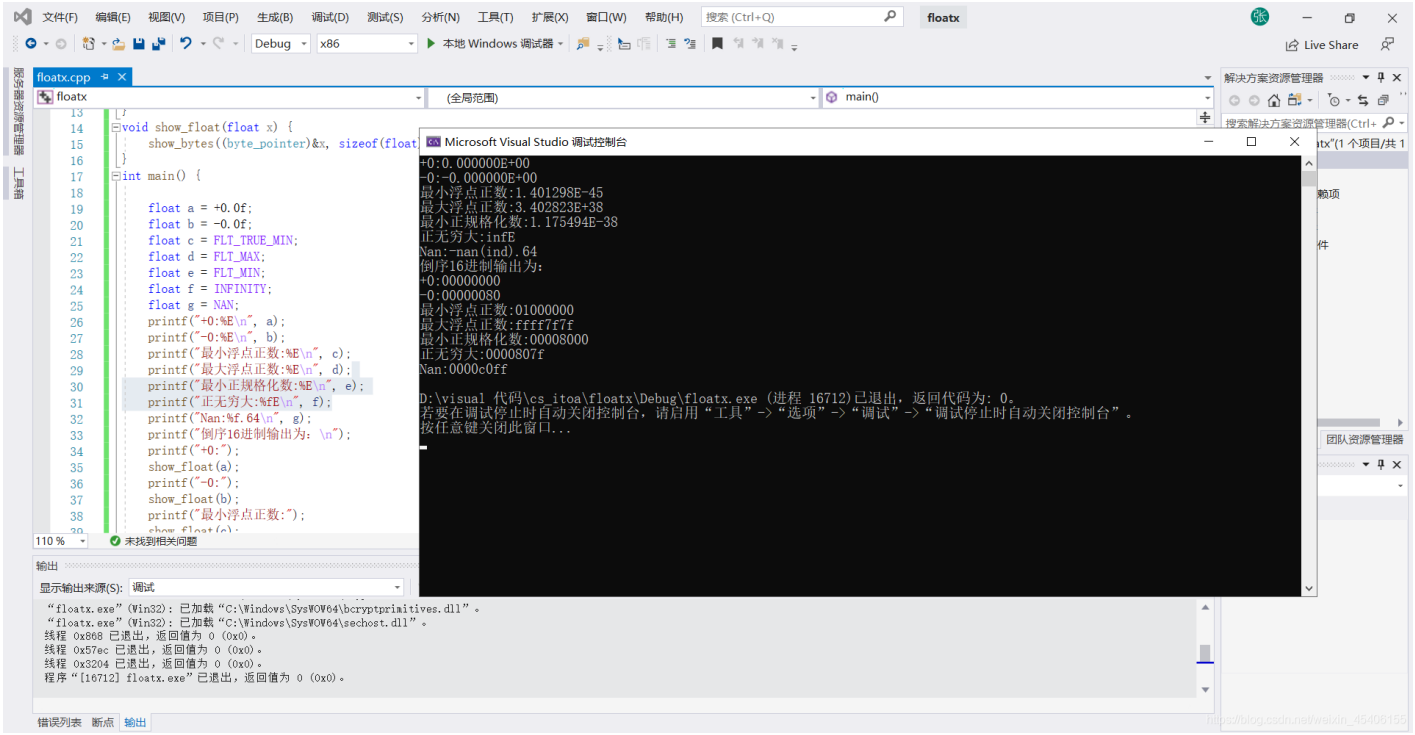
提交子程序floatx.c，要求：

```
#include<stdio.h>
#include<stdlib.h>
#include<float.h>
#include <math.h>
#define _GNU_SOURCE 1
typedef unsigned char* byte_pointer;
void show_bytes(byte_pointer start, size_t len) {
 size_t i;
 for (i = 0; i < len; i++) {
  printf("%.2x", start[i]);
```

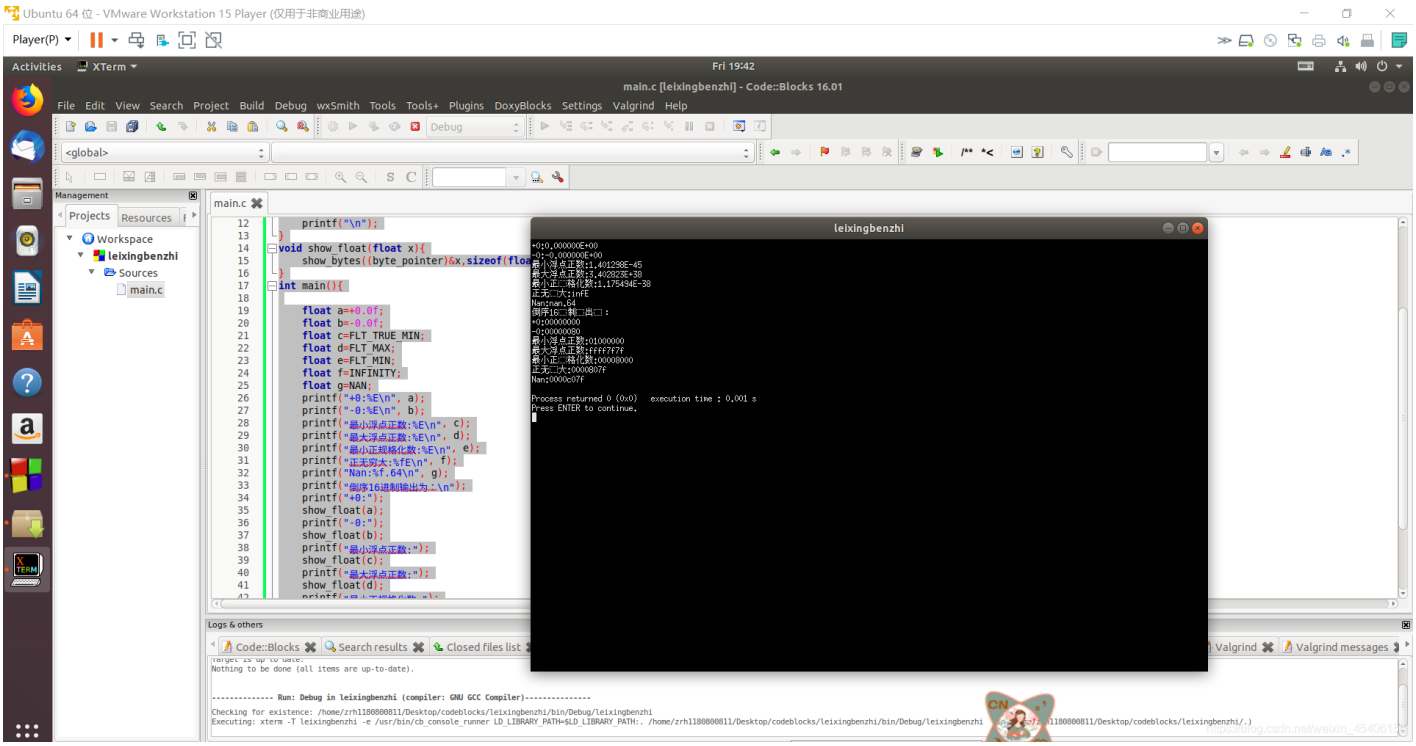
```
 }
 printf("\n");
}
void show_float(float x) {
 show_bytes((byte_pointer)&x, sizeof(float));
}
int main() {

 float a = +0.0f;
 float b = -0.0f;
 float c = FLT_TRUE_MIN;
 float d = FLT_MAX;
 float e = FLT_MIN;
 float f = INFINITY;
 float g = NAN;
 printf("+0:%E\n", a);
 printf("-0:%E\n", b);
 printf("最小浮点正数:%E\n", c);
 printf("最大浮点正数:%E\n", d);
 printf("最小正规格化数:%E\n", e);
 printf("正无穷大:%fE\n", f);
 printf("Nan:%f.64\n", g);
 printf("倒序16进制输出为：\n");
 printf("+0:");
 show_float(a);
 printf("-0:");
 show_float(b);
 printf("最小浮点正数:");
 show_float(c);
 printf("最大浮点正数:");
 show_float(d);
 printf("最小正规格化数:");
 show_float(e);
 printf("正无穷大:");
 show_float(f);
 printf("Nan:");
 show_float(g);

}
// 运行程序: Ctrl + F5 或调试 >“开始执行(不调试)”菜单
// 调试程序: F5 或调试 >“开始调试”菜单

// 入门使用技巧:
//   1. 使用解决方案资源管理器窗口添加/管理文件
//   2. 使用团队资源管理器窗口连接到源代码管理
//   3. 使用输出窗口查看生成输出和其他消息
//   4. 使用错误列表窗口查看错误
//   5. 转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项”以将现有代码文件添加到项目
//   6. 将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件
```

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。

一句两句说不明白，结果大概这样。



Visual stdio 下输出



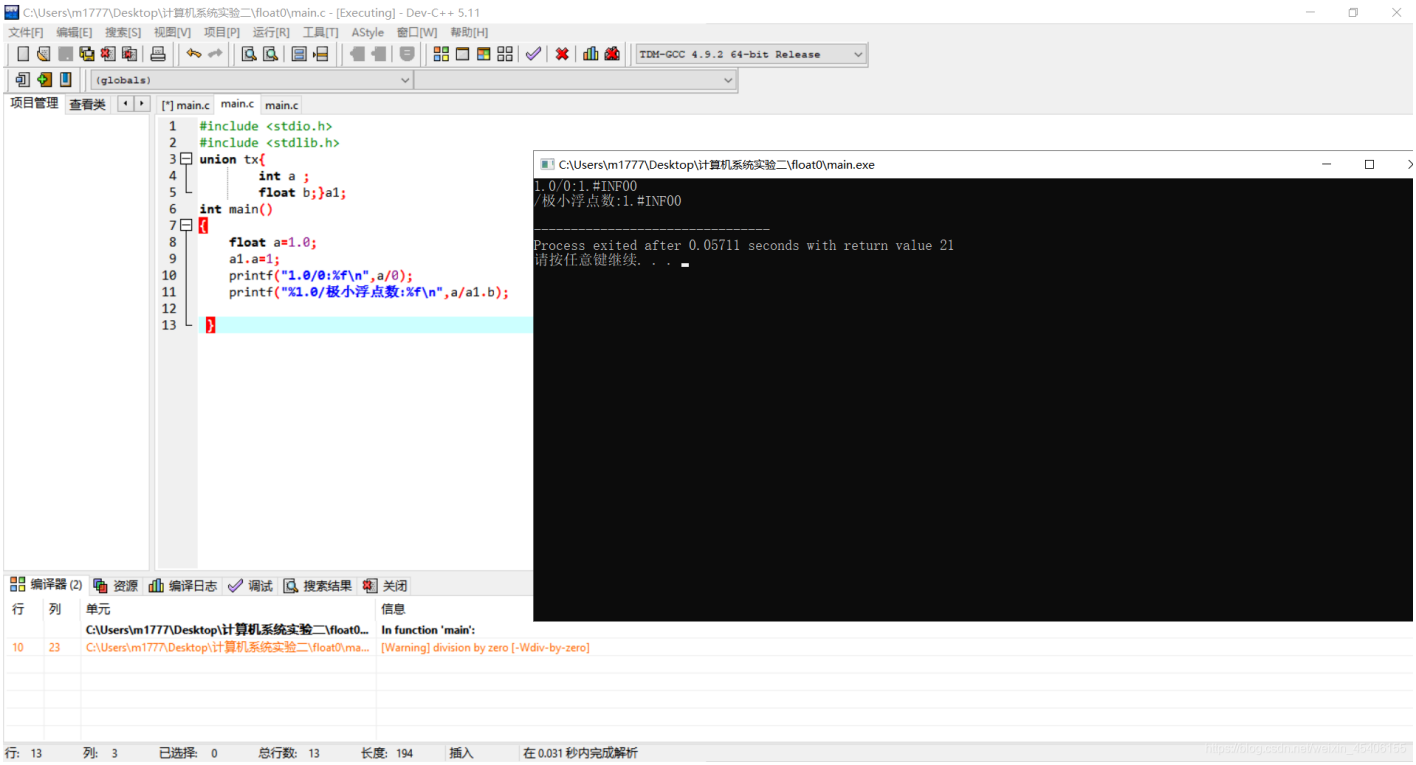
Linux CB 输出

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

Float除零和最小非规格化数均为正无穷，除一个小浮点数会得到一个很大的数。

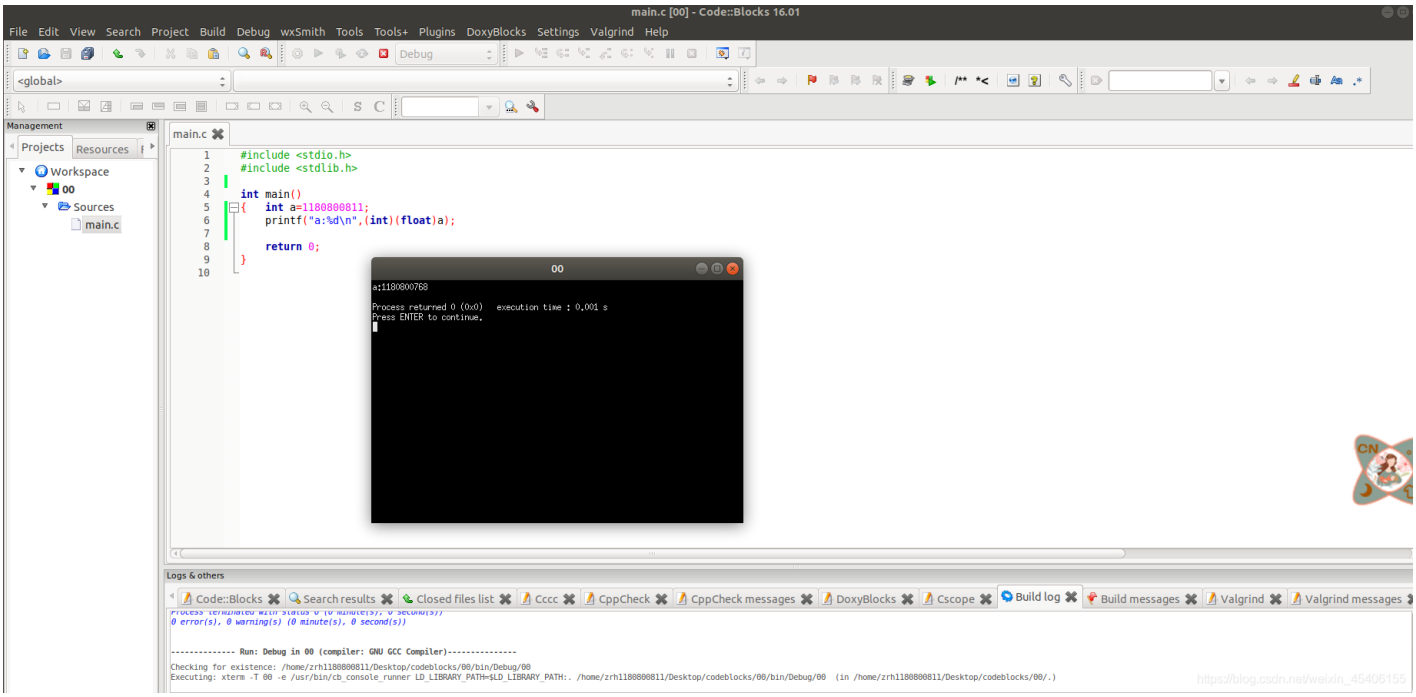


7.4 类型转换的坑

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=__1180800811__，现x=_1180800768_____

这个就直接按步骤就行了，结果因人而异



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

_____

7.6 讨论2：怎么验证float采用的向偶数舍入呢

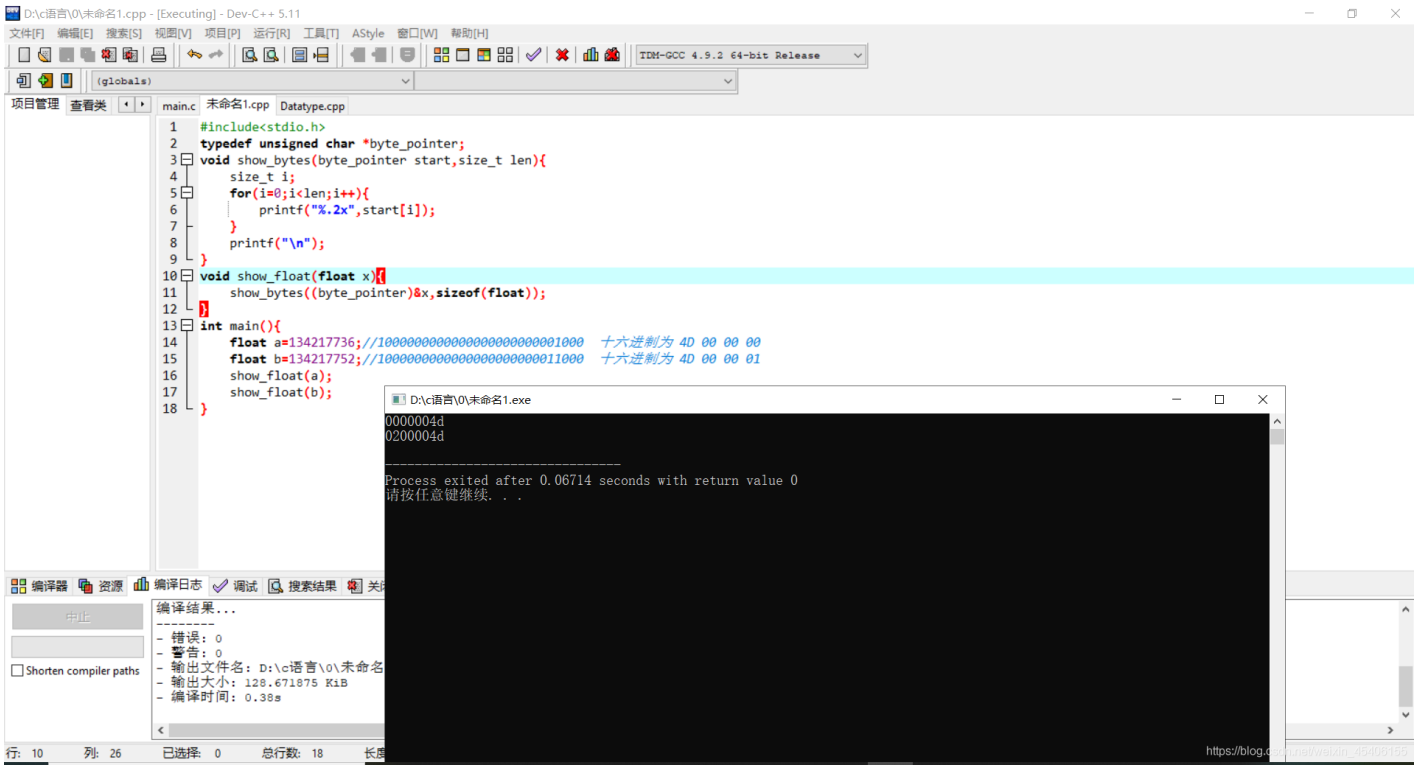
基于上个讨论，开发程序或举几个特例用C验证即可！

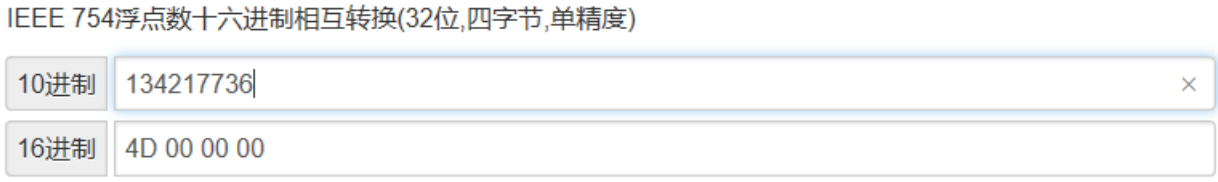
截图与标注说明！

找两个数字， 第一个数字第一位是1，第24位是0，第25位是1，25位后全为0，其他位随意。

第二个数字第一位是1，第24位是1，第25位是1,25位后全为0，其他位随

意。





IEEE 754浮点数十六进制相互转换(32位,四字节,单精度)

10进制	134217752
16进制	4D 00 00 01

可以看出，第一个数字第24位是0，第25位是1，由于float类型尾数只有23位，所以需要进行舍入，由于第24位是,0，而第25位是1，根据向偶数舍入的原则，要使第24位为偶数位，此时第25位向低位舍入，因此打印出来的值为4D 00 00 00

而第二个数字第24位是1，第25位是1，根据向偶数舍入的原则，要使第24位为偶数位，此时第25位向高位舍入，因此第二个数字比原来数字大16，16进制表示为1，因此源数字由4D 00 00 01 变为4D 00 00 02。

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 0.50.75

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）：_-(2-2^-23)*2^-126~(2-2^-23)*2^-126_____、_____2^-149_____、___-(2-2^-23)*2^127~2^127　　和　2^127~(2-2^-23)*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____1.401298*10^-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___2.028241*10^31_____

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。

因为计算机存储的特性，任意两个浮点是不能用==直接比较

比较好的方法就是用两个数之间的差值小于某个最小值，

比如对于两个浮点数a,b，如果要比较大小，那么常常会设置一个精度
如果fabs(a-b)<=1e-6，那么就是相等了。 fabs是求浮点数绝对值的函数。
类似的 判断大于的时候，就是if(a>b && fabs(a-b)>1e-6)。

判断小于的时候，就是if(a<b && fabs(a-b)>1e-6)。

对于以及转化为IEEE型的浮点数：

1. 首先判断是否为NAN，若为两个数都为NAN，vb中默认相等，cb中不可比较，
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，则解码小的数小
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

计量单位的改变，常常出现数据不平衡的现象

在报表处理过程中，有时需要对数据进行舍位平衡。舍位，就是将表格中的数据抹去尾数的一位或几位数，例如：将以元为单位的表格，改为以千元为单位的数据表格，就是舍去表格中数据的后三位。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。

8.2 给出完美的解决方案

首先：提高数据精度，float换double=〉long double，要求提高上报数据精度，原来整数，现要求小数后2位，原来2位，现要求5位等。
除了上报统计数据，还要上报明细数据。这也是信息系统建设要求从区县级集中到市级集中，再到省级集中，再到全国大集中的发展现状。

再者，可用以下模型解决：

有一组报表数据，如表所示。

其中:aij为实数 (i=1, 2, …, m;j=1, 2, …, n) 。

a1k=∑i=2maik(k=1,⋯,n)

a11=∑j=2nalj(l=1,⋯,m)

通过调整除a11以外的其他任意数据aij(i,j不同时为1），使得

表1平衡数据

a_{11}	a_{12}	……	a_{1n}
a_{21}	a_{22}	……	a_{1n}
……	……	……	……
a_{m1}	a_{m2}	……	a_{mn}

a1k=∑i=2maik(k=1,⋯,n)，a11=∑j=2nalj(l=1,⋯,m)

其中, [aij]表示对数据aij进行四舍五入后取整数。上面的问题即为舍位平衡问题。

（1）舍位平衡条件

能使任意两个不平衡的向列 (平衡差均不为0) , 调整为平衡的充分条件有:①两个不同方向的子向列的平衡差符号相同;②两个相同方向的子向列的平衡差符号相异;③和向列与不同方向的子向列的平衡差符号相异;④和向列与相同方向的子向列的平衡差符号相同;⑤两个和向列的平衡差符号相同。

由此可见, 要想使一个平衡单元上的数据平衡, 就要找出符合平衡条件的两个不平衡向列来进行调整。通常, 在一个平衡单元内, 会发生同时满足多个条件的情况, 这就要逐个条件的分别进行调整。

一般地, 在一个平衡单元中, 只要有一个向列发生不平衡 (平衡差不为0) , 则必然存在有另一个向列也不平衡, 而且这两个向列满足上面的平衡条件。

（2）舍位平衡算法

只要针对以上5个平衡条件, 分别进行相应的调整, 即可形成平衡算法。

首先定义调整值:

$$D = \frac{\text{平衡差}}{|\text{平衡差}|}(\text{即}\pm 1)$$

对应于5个平衡条件的舍位平衡算法的描述为①在两个子向列的交叉项上, 加上D;②取其中一子向列 (任意) 的D值, 在这两个子向列的非合计项上, 并行查找不为0的两个数, 并且这两个数所在的子向列(另一方向) 上的平衡差为0, 对其中一个数 (取D值的子向列上的那个) 加D, 对另一个数减D;③在两个向列的交叉项上, 加上和向列的D值;④在这两个向列的非合计项上, 并行查找不为0的两个数, 并且这两个数所在的子向列 (另一方向) 上的平衡差为0, 然后对这两个数分别加上D;⑤找到一个不在和向列上的任意不为0的数, 并且这个数对应交叉位置的两个子向列上的平衡差为0, 然后对这个数以及对应交叉位置的两个子向列上的平衡差为0, 然后对这个数以及对应交叉的两个子向列上的合计项 (两个) , 分别加上D。

在以上的舍位平衡算法中, 每执行一次某个算法作业后, 其两个向列仍然满足相应的平衡条件 (平衡差均不为0) , 这时, 应再执行一次同样的算法作业, 一直到这两个向列不满足相应的平衡条件为止。

平衡算法的优先级为①->②->③->④->⑤

（3）舍位平衡的几个问题的讨论

1、双向、单向平衡

上述算法所讨论的是有两个和向列的平衡问题, 即双向平衡。还有另一种情况, 就是只有一个和向列的平衡问题, 即单向平衡。其平衡算法较为简单, 因为在作单向平衡时, 不用考虑对另一个方向上的向列的影响。

对单向平衡的平衡算法为①在向列上的非合计项中, 找到一个不为0的数;②将这个数减去D, 向列上的平衡差减一次D;③如果平衡差仍不为0, 继续找下一不为0的数, 返回第二步, 否则结束。

1. 双向平衡中的单向平衡处理

有时会出现这样的情况:数据式A中, 有个别子向列不属于某一方向的和关系式中, 同时也不属于其他任何平衡单元的同一方向的和关系式, 但这些向列本身又满足另一方向的和关系式, 即这些向列对向是独立的, 而对另一向则是非独立的。

对这类向列的平衡调整, 实际上已经演变为单向平衡问题。通常在进行双向平衡时, 所有涉及的向列 (报表中即行、列) , 都给出一个标志, 那些没有给过标志的, 便是这类单向平衡处理的向列。

1. 平衡优先层级

上面所述的是一个平衡单元的简单平衡问题, 对于多个平衡单元, 而且具有嵌套的层级关系的平衡问题, 处理方法如下：

对一个单元平衡时, 有一个基本原则:就是总合计数 (即数据式A中的al1) 是不能动的, 但可以调整其他数。如果是由内层往外层作平衡, 那么, 内层平衡了, 内层的和向列又要作为外层的子向列, 在对外层平衡时, 可能会调整到这些子向列 (内层中的和向列) , 这样又破坏了内层原来的平衡。因此, 平衡的优先层级为由外层到内层。

4、关系式中的减运算处理

上面讨论了和运算 (+) 的平衡问题, 即和向列等于子向列之和, 至于关系式中出现减运算的情况, 可以将减运算化为加运算, 即将被减的向列乘上一个因子-1, 待平衡处理后再还原回来 (再乘上因子-1)即可。

（3）对一个数据单元的舍位平衡过程举例

通过下面的一个平衡例子简单说明对一个单元的平衡过程。有一个数据单元:见图所示。其中:行1=行2+行3;列1=列2+列3+列4, 现在要将这组数据按照四舍五入舍去最后1位, 化整后结果见图2所示。

行列 →	(1)	(2)	(3)	(4)
(1)	217	65	97	55
(2)	142	26	74	42
(3)	75	39	23	13

图1 数据单元

平衡差 →		+1	+1
-1	22	7	10	6
	14	3	7	4
+1	8	4	2	1

图2 数据单元四舍五入化整处理

平衡步骤:(1) 用平衡算法①, 将第3行、第3列交叉上的数2调整 (加1) 得3。

1. 用平衡算法③, 将第1行、第4列交叉上的数6调整 (减1) 得5。

这样, 平衡后的结果见图所示。

22	7	10	5
14	3	7	4
8	4	3	1

图3 舍位平衡后的结果

第9章 总结

9.1 请总结本次实验的收获

首先，对于Linux的命令集有了更好地认识，能更熟悉的操作Linux系统，同时对汇编代码也有了更深的熟悉，对机器级指令有了更深的认识。同时对浮点数在机器内的表示有了更深的认识，从无知到略懂，计算机系统试验教我重新认识到自己的不足，自己的无能之处。

9.2 请给出对本次实验内容的建议

老师能不能提前几天吧每次试验的内容发下来，让我们能做好充分的预习，这样在上实验课的时候也能事半功倍。同样，对于一些特别难的题老师能不能给点提示，这样就不会一点头绪都没有反而浪费了很多时间。希望老师把培养方案变得简单一点吧，才大二呢，就这么难。还有那个万年虫的，真的做不出来，老师也不给点提示。

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.