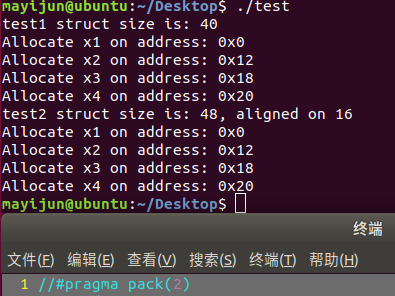
作业一 数据的机器级表示与处理

马逸君 17300180070

第一题 数据对齐、\_\_attribute\_\_(aligned())、#pragma pack()

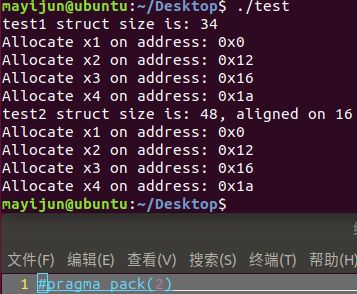
默认设置

test1结构体占用的存储空间大小为40字节，

x1、x2、x3、x4相对结构起始位置的偏移量分别为0、18、24、32。

这是因为默认设置下采用了自然对齐方式，char、short、int、long long分别按1字节、2字节、4字节、8字节对齐。

test2结构体占用的存储空间大小为48字节，x1、x2、x3、x4相对结构起始位置的偏移量分别为0、18、24、32。结构体占用空间与test1不同的原因是，为了使连续申请存储空间时每一个test2类型的对象都可以按16字节对齐，分配空间时会在每一个test2对象的尾部插空，使得占用字节是16的整数倍。

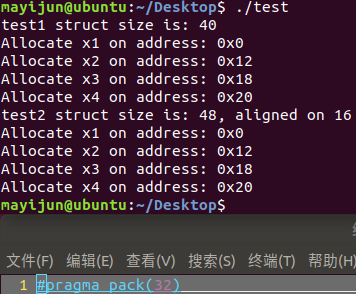


#pragma pack(2)

test1结构体占用的存储空间大小为34字节，x1、x2、x3、x4相对结构起始位置的偏移量分别为0、18、22、26。

test2结构体占用的存储空间大小为48字节，x1、x2、x3、x4相对结构起始位置的偏移量分别为0、18、22、26。

这是因为#pragma pack(2)语句指定结构体内部成员按照min(自然边界,2byte)对齐，所以除char仍按1字节对齐外，各成员变量都按2字节对齐。

#pragma pack(32)

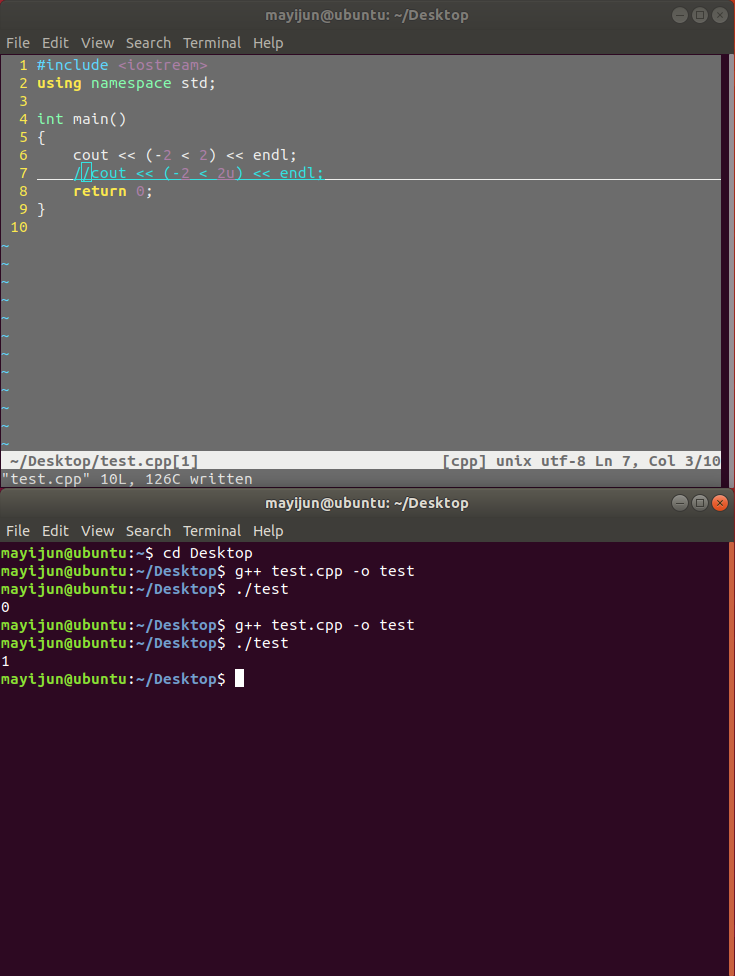
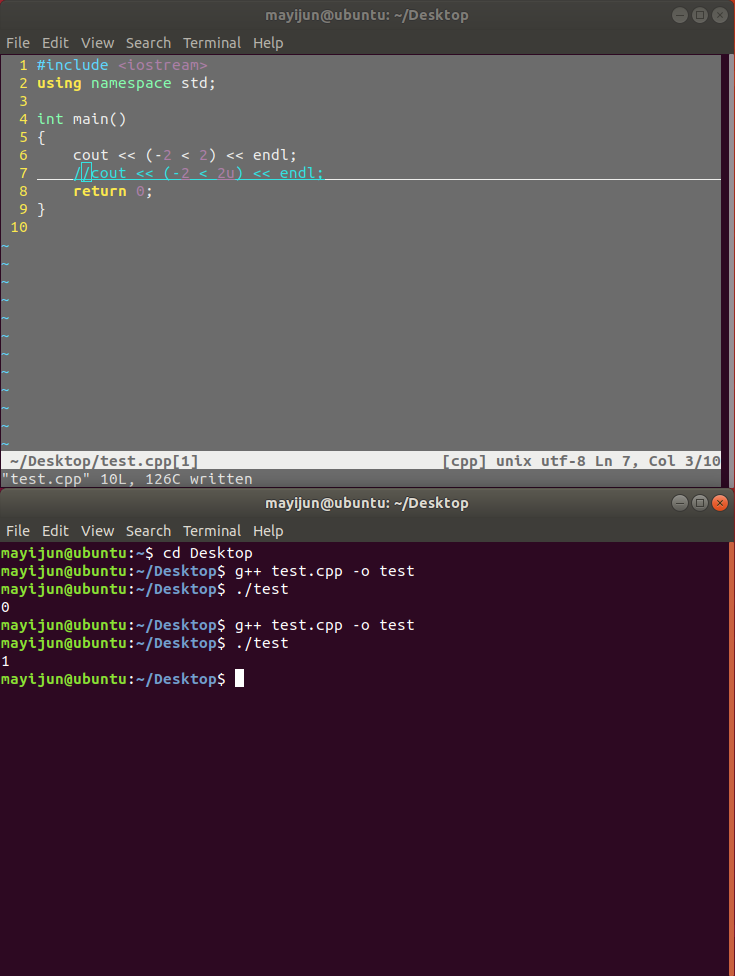
test1结构体占用的存储空间大小为40字节，x1、x2、x3、x4相对结构起始位置的偏移量分别为0、18、24、32。

test2结构体占用的存储空间大小为48字节，x1、x2、x3、x4相对结构起始位置的偏移量分别为0、18、24、32。

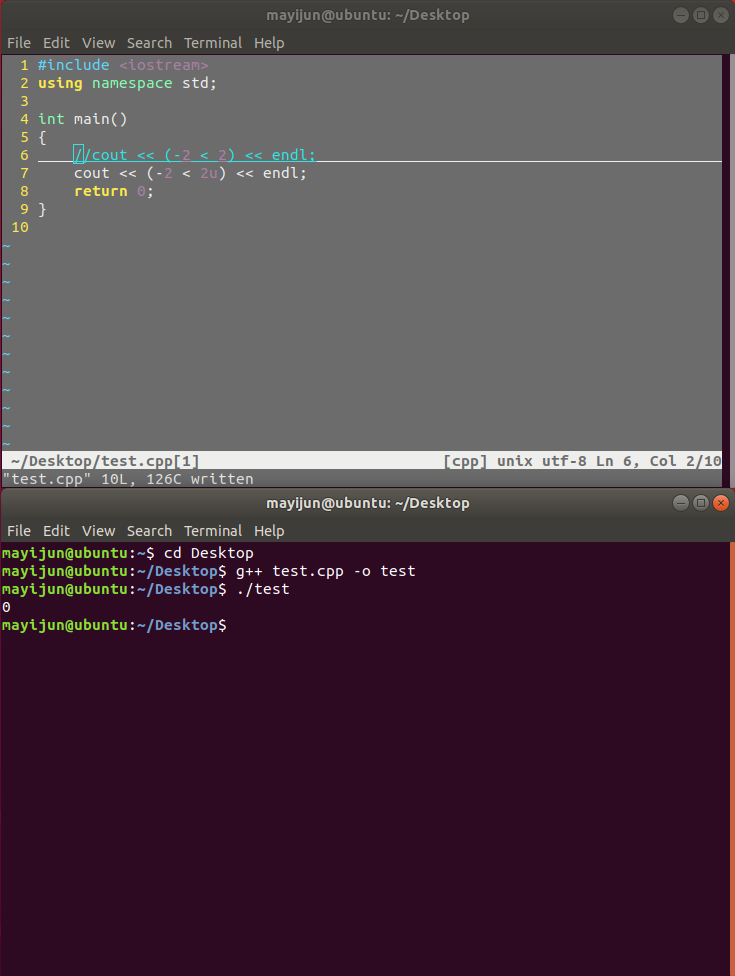
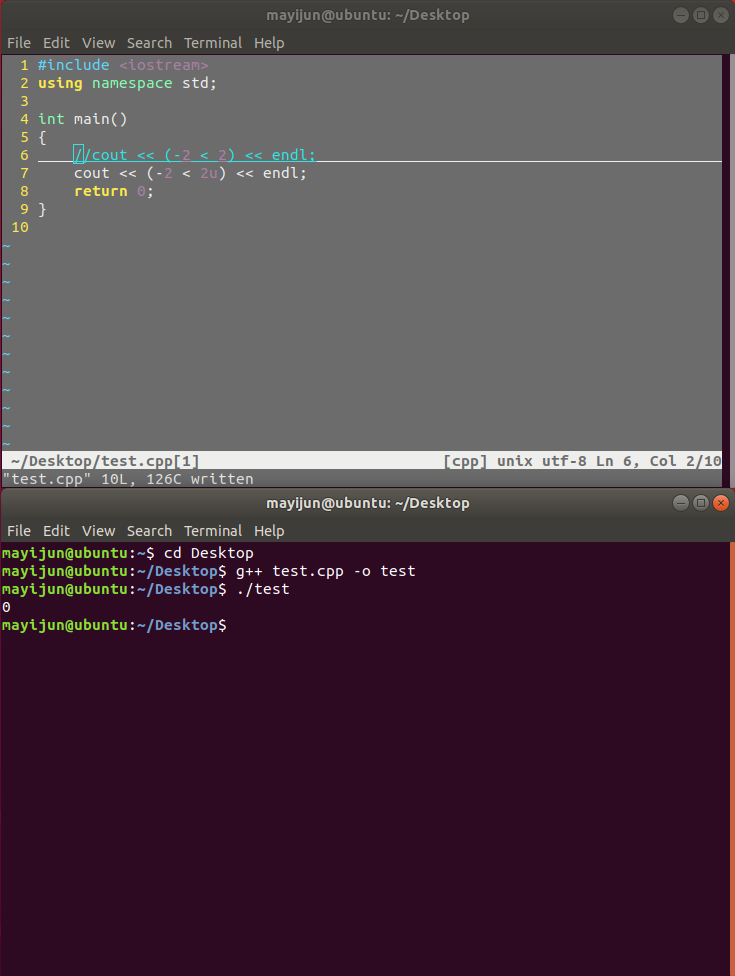
#pragma pack(32)语句指定结构体内部成员按照min(自然边界,32byte)对齐，但是因为test1和test2各成员变量的自然边界都小于32byte，所以它们仍然是按自然边界对齐的。

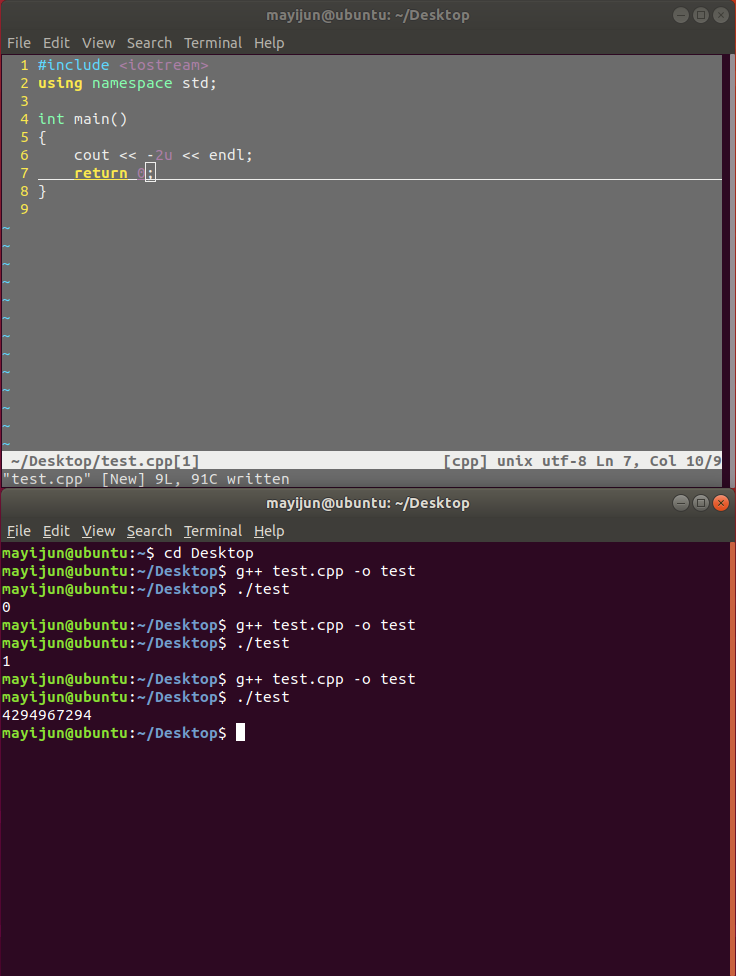
第二题 强制类型转换：有符号转无符号

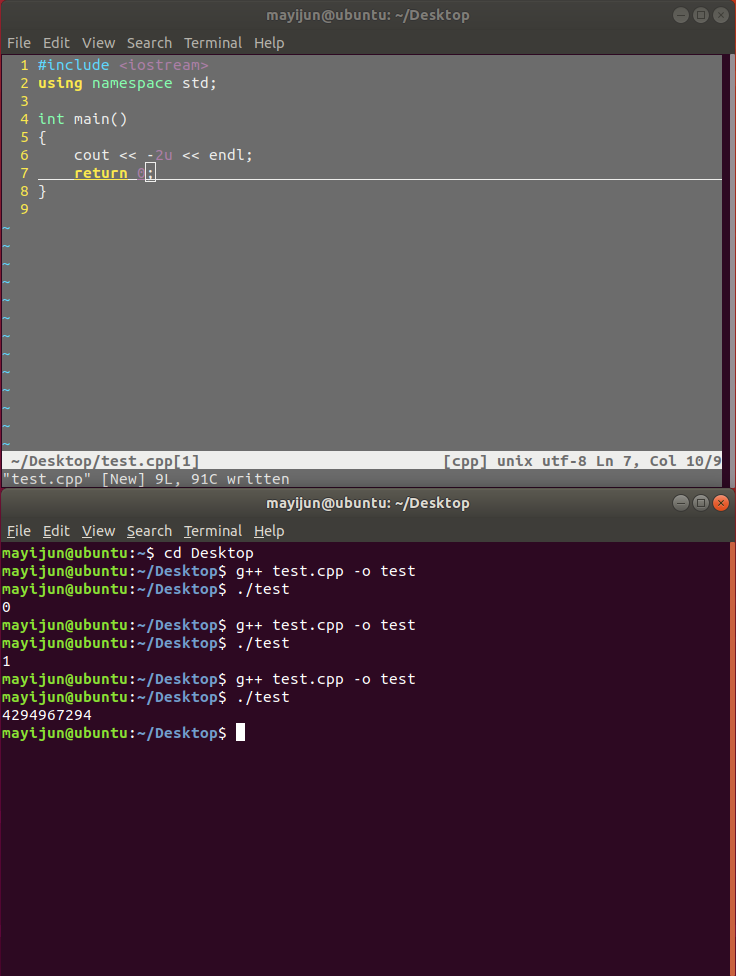
-2 < 2的结果为：true，正确性显然



-2 < 2u的结果为：false

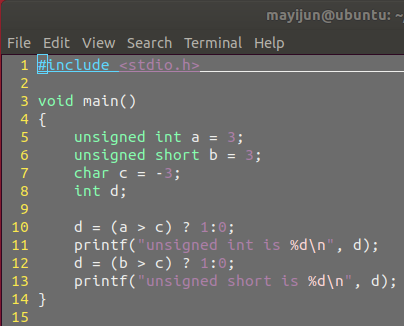
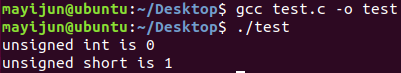


其原因是，当参与比较的两个操作数类型不同时，低类型会自动向高类型转换，与int相比，unsigned是高类型，所以-2会先被强制转换为unsigned类型的-2u（其值为232-2），再与2比较，故结果为大于。

附：C++数据类型级别从高类型到低类型依次为：

long double、double、float、unsigned long long、long long、unsigned long、long、unsigned int、int（但当long和int宽度相同时，unsigned int级别高于long）

第三题 强制类型转换：整型提升

运行结果如图。

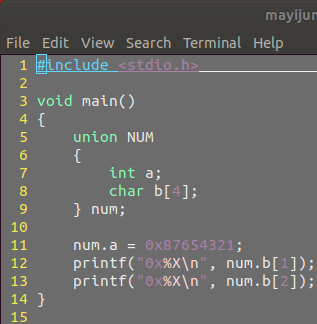
这一题涉及到一个概念叫“整型提升”(integer promotion)1234：

C的整型算术运算总是至少以缺省整型类型的精度来进行的。为了获得这个精度，表达式中的字符和短整型操作数在使用之前被转换为普通整型，这种转换称为“整型提升”。

有符号或无符号的char、short在做算术运算之前，首先要做整型提升（即自动转换为int类型），然后才能参与运算（此时可能会进一步转换成更高类型）。

所以，第一小题的-3自动转换成了unsigned int类型的-3u（值为232-3），故结果为小于；第二小题的unsigned short和char被自动转换为了int型的3和-3，故结果为大于。

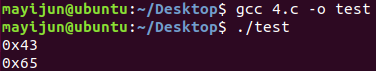
（另外值得一提的是，int作为缺省整型类型，是计算机处理效率最高的长度。如果不是必须选择其他类型，最好使用int。）

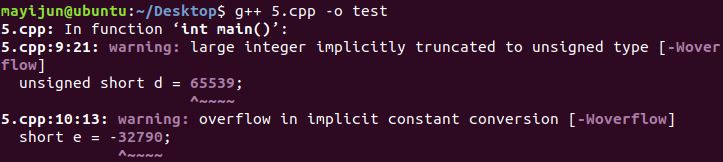
第四题 union

实验结果表明，对union中的一个成员赋值，另一个成员也会被改变。

这是因为，union的存放顺序是所有成员开始于同一地址。union使得程序员可以方便地通过不同方式使用同一个内存位置。

我们看到，此例中char数组b[4]和int变量a指向完全相同的一片地址；从LSB开始计数，b[1]的值与a的第二位相同，b[2]的值与a的第三位相同。由此可得a的LSB在低地址，MSB在高地址；本机的CPU存放数据的方式是小端方式。



第五题 机器数与真值

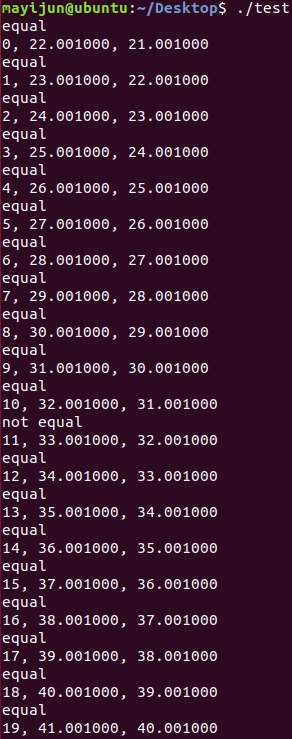
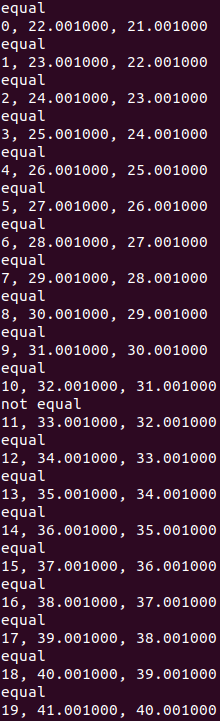
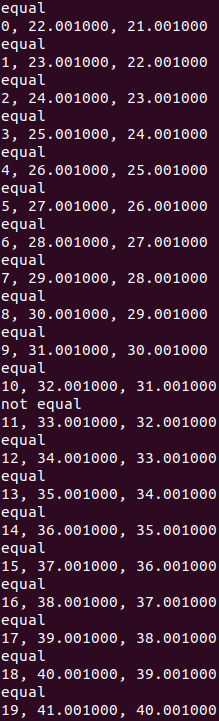
|  |  |  |
| --- | --- | --- |
| 变量 | 机器数 | 真值 |
| a | 1000 0000 0000 0000 0000 0000 0000 0000 | -2147483648 |
| b | 1000 0000 0000 0000 0000 0000 0000 0000 | -2147483648 |
| c | 1000 0000 0000 0000 0000 0000 0000 0001 | -2147483647 |
| d | 0000 0000 0000 0003 | 3 |
| e | 0111 1111 1110 1010 | 32746 |

其中d和e所赋的值超出了16bit所能表示的范围，所以会强行截断，其机器数为截断后保留的低16位的信息。然后再按它们的数据类型推算真值。

编译器出现warning的原因也在于超出范围、发生了截断。

第六题 计算2n的浮点数表示

见所附源代码。

第七题 浮点误差

运行结果如上所示。三次运行的结果是一致的。

分析“not equal”出现的原因：

20.001写成二进制小数是：10100.00000000010000011001

20.001的浮点数表示为：0 1000 0011 0100000000000100000110**1**

尾数第23位以后的部分为“1”，此时适用保留偶数原则，故进位。

31.001写成二进制小数是：11111.00000000010000011001

31.001的浮点数表示为：0 10000011 1111000000000100000110**1**

31.001是从最初赋值的20.001逐次加1而来的，阶码未变，未发生舍入。

32.001写成二进制小数是：100000.00000000010000011001

32.001的浮点数表示为：0 10000100 0000000000000010000011**0**

32.001是从31.001加1而来的，运算时发生了右规，需要舍入。此时适用保留偶数原则，故舍去。

这样一来，相减时就多减了2-19，根据浮点加减法的运算过程，32.001-31.001的结果就为  
0 01111111 111111111111111111100000，不等于1.0。

因为只有32.001-31.001时会有上述问题，所以三次运行时都有且仅有这一种情况是not equal。

但在每步循环输出时，输出结果的小数部分仍然都是.001，这是输出时自动保留6位小数的结果：机器数和精确值的误差仅在2-19级别，输出时这个误差被忽略，所以输出仍为.001。

参考：

第三题 强制类型转换：整型提升

<https://blog.csdn.net/wenhai_zh/article/details/8156089>

<https://baike.baidu.com/item/整型提升/16697764?fr=aladdin>

<https://blog.csdn.net/ZWE7616175/article/details/76088640>

<https://blog.csdn.net/u014190698/article/details/80458701>

<https://blog.csdn.net/qq_35524916/article/details/61409126>

第四题 union

<http://c.biancheng.net/view/375.html>