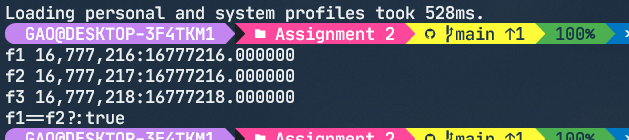
**浮点数表示方法报告**

**18081811-高炜哲**

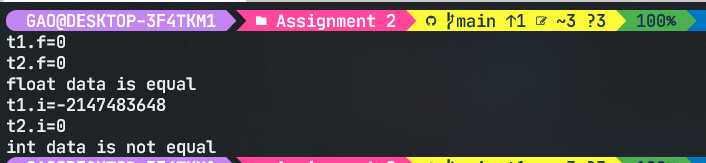
f1 f2 f3 实际上是3个不一样但是十分接近的浮点数，由于计算机的浮点数为**离散量**且表示**精度不足**，所以根据“**就近原则**“，f2无法准确表示，就近表示为f1，所以f1，f2打印均为217:16777216.000000，而f3可以准确表示为217:16777218.000000。



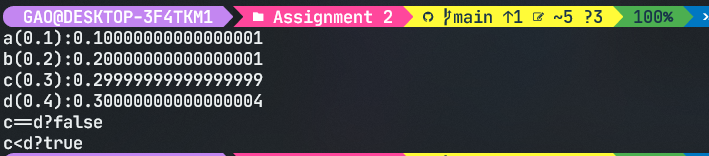
a的**除数为0**，而float, double中存在一类特别的数：**-1.#IND**，IND其实就是Implementation Not define，这个是**未定义**或者他不是一个数。同理，b为-1的开方，**1.QNAN0**其实就是NaN, 即**Not a Number**，因此他同样不是一个数。



结构体中的float f均不赋值，默认为0，所以t1.f == t2.f；t1.i = 0x80000000，二进制补码为1000 0000 0000 0000 0000 0000 0000 0000，为**最小负数-2147483648**，而**t2.i = 0x00000000 = 0**，所以两者并不相等。

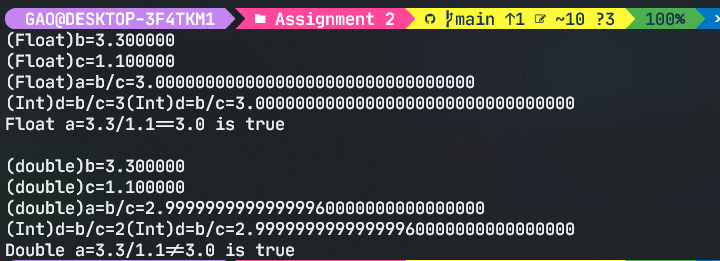


以0.1为例，0.1表示为二进制时实际上为0.0001100110011…**无限循环小数**，**截断**并将其转化为10进制数为0.1000000000000000055511151231257827021181583404541015625，与其最接近的17位小数即为0.10000000000000001。b, c, d 三个小数同理，显然c不等于d且c < d。



C语言中，float是**单精度浮点数**，而double是**双精度浮点数**，所以Double的精度是Float的两倍。所以由于精度不同的原因，我将d的值分别以double格式输出，可见在float下，**3/1.1=3.000…**，而double下，**3/1.1=2.999…6**，所以变作int时小数点后面截断，分别位3和2。

对于d同理，float下为3.000…，double下为2.999…6, 所以3/1.1在float下等于3，在double精度下则不等于。



(3.14+1e20)-1e20 : 由于1e20远远大于3.14，因此前两者相加时**精度不足**，所以**3.14+1e20被记为1e20**，所以1e20 – 1e20 = 0；

3.14+(1e20-1e20) : 后两者先相减为0，3.13 + 0 = 3.14。



改程序主要是测试两数**相加上溢**和两数**相减**所出现的情况。

前3行是输出内存地址，t1.f和t2.f值为该内存地址7F0000000里的数值，相加时候结果**正溢。**

同理，两数相减时可被正常表示。

