“计算机组织结构”作业 02

1. 下列几种情况所能表示的数的范围是什么？
   1. 16 位无符号整数
      - 0 ~ 2^16 -1，即 0 ~ 65535
   2. 16 位原码定点小数
      - -(1-2^-15) ~ 1-2^-15
   3. 16 位补码定点整数
      - -2^15 ~ 2^15 - 1，即-32768 ~ 32767
2. 设某浮点数格式为：1 位数符、5 位阶码、6 位尾数。参照 IEEE754 浮点数的解释方式，写出：
   1. 规格化数的非零正数的最小值、最大值
      - IEEE754 为：2^-126 ~ (2-2^-23)*2^127
      - 2^-14 ~ (2-2^-6)*2^15
   2. 非规格化数的最小值、最大值
      - IEEE754 为：2^-(126+23) ~ (1-2^-23)*2^-126
      - 2^-(14+6) ~ (1-2^-6)*2^-14
   3. 写出 9/16 的二进制表示。
      - 0011 0000 1000B
      - 9/16 = 0.1001B = 1.001 * 2^(-1)，符号位为 0，指数为-1，5 位阶码表示的指数偏置常数为 2^(5-1)-1 = 15，故移码表示为 -1 + 15 = 14 = 0 1100B，尾数部分为 00 1000B。
3. 假定变量 int i = 1234567890、float f = 1.23456789e9，sizeof(int)=4，判断以下表达式的结果（True / False）
   1. i == (int)(float)f；i == (int)(double)f
      - False；False。
      - 第一个表达式：因为 IEEE754 的 float 类型中，尾数的小数部分只有 23 个二进位和一位隐藏位，共 24 位有效位数，理论上 float 的十进制有效位数为 7 位，而 i 中有 9 位十进制有效位，用二进制表示为 1110101101111111000000111 110B，从二进制有效值的编码来看，将 i 转换为 float 类型时会发生 3 位有效数字的丢失，再转换为 int 类型时，其值无法还原了
      - 第二个表达式：因为 IEEE754 的 double 类型中，尾数的小数部分有效位数为

52+1=53 个二进位，而 int 类型的有效位数只有 31 个二进位，因此对于任何一个 int 类型的变量，转换为 double 后，其精度不会有损失，但是由于最开始赋值时的 float 损失了精度，在二进制中已经舍弃了后续的 3 位有效值，double 是无法还原精度的

2. i == (float)(int)f；i == (float)(double)f

- False；False。
- 第一个表达式：当 float 变量的值在 int 可表示范围内时，由于 float 的尾数小于 int 的位数，先转换为 int 再转换回 float 不会导致精度损失；当超出 int 范围时(如 i=f=1.23456789e10)，则数值无法正常转换。
- 第二个表达式：double 类型的有效值位数比 float 类型多，任何 float 类型的变量转换为 double 后再转换回 float 类型时，其值不变
- 从结果上说，两个表达式中都因为 float 类型在最初赋值时就已经损失了精度，因此无论是使用 int 还是 double 进行转换，都无法还原回与 int 类型的变量 i 相同的值

4. 下图是某个 java 程序及其该程序的若干组执行结果。请根据 IEEE754 标准的舍入规定对运行结果进行解释说明，并通过分析得出 float 变量的有效十进制位数

```
public static void main(String[] args) {
    float f;
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    while (true) {
        System.out.print("please enter a number: ");
        try {
            f = Float.parseFloat(br.readLine());
            System.out.printf("%f\n", f);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
please enter a number: 61.419997
61.419998
please enter a number: 61.419998
61.419998
please enter a number: 61.419999
61.419998
please enter a number: 61.42
61.419998
please enter a number: 61.420001
61.420002
please enter a number: 
```

- 7 位
- 该程序的功能非常简单，就是从键盘上输人一个实数，赋给一个 float 型变量后再从屏幕上输出。从运行结果来看，61.419998 和 61.420002 是两个可表示数，两者之间相差 0.000004。当输入数据是一个不可表示数时，机器将其转换为最邻近的可表示数。
- 目前几乎所有机器中 float 型变量都是采用 IEEE754 单精度浮点数格式表示，其二进制有效位数为 24 位，因此能精确表示的十进制有效位数为 7 位。因为 61 = 111101B = 1.11101B×25，如果将 float 型数据的规格化正数的表示范围以 2^i (-126≤i≤127) 为分割点划分成若干区间，61.419998 应该位于区间 [2^5，2^6]，该区间相邻可表示数之间的间隔为 2^-23 × 2^5 = 2^-18 = 0.0000038…≈ 0.000004，从上述分析结果来看，该区间相邻两个可表示数之间的间隔就是 0.000004。因此，在 61.419998 前面的可表示数为 61.419994，后面的可表示数为 61.420002。
- 从直观的角度看，输入 61.419997 和 61.419999 时，其最靠近的可表示数为

61.419998；而 61.420001 的最邻近可表示数为 61.420002

- 但在特殊的中点值情况下，如输入为 61.42 时，十进制形式的 61.420000 位于可表示数 61.419998 和 61.420002 的中点。这时与机器的舍入机制有关，需要根据机器内部的二进制表示来判断
- 61.42 的 double 表示为 1 10000000100 1110101101011100001010001111010111000010100011110110
- 从二进制表示形式来看，在 float 能表示的有效值部分的后一位为 0，因此后续数字被舍弃，因此，61.42 对应输出的可表示数为 61.419998。
- 实际上，浮点数所有的可表示数都是根据二进制来选择的，因此中点值的可表示数需要根据实际情况来判断。

================== 分割线：以下内容不在小程序上提交 ==================

5. 设一个变量的值为 2049，在程序中将其转换为 32 位整数补码、IEEE754 单精度浮点数格式并打印该变量（用二进制字符串表示），找出两种编码中表示有效值的二进制序列，并说明这段序列不同的原因及浮点数表示有效值的方式的优势
    - 2049 = 1000 0000 0001B = +1.000 0000 0001B × 2^11
    - 32 位补码整数表示为 0000 0000 0000 0000 0000 1**000 0000 0001**B
    - IEEE754 单精度浮点数格式表示为 0100 0101 0**000 0000 0001** 0000 0000 0000B
    - 上述两种表示中，存在相同的二进制序列 000 0000 0001。因为 2049 被转换为规格化浮点数后，有效值部分中最前面的 1 被隐藏，而 32 位补码整数中最前面的 1 没有被隐藏，所以除了这个 1 之后的二进制有效值序列是相同的。
    - 这意味着浮点数的尾数可表示的位数比实际编码多一位，因而使浮点数在相同有效值位数的情况下精度能够更高