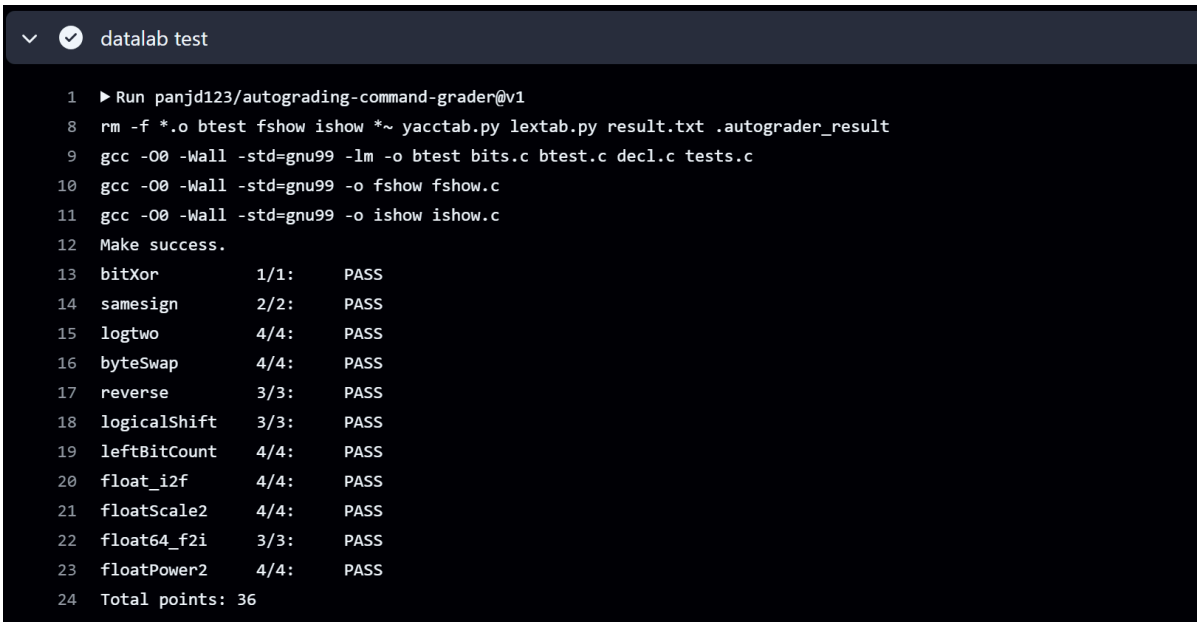
datalab 报告

姓名：卢虹宇

学号：2023202269

总分	bitXor	logtwo	byteSwap	reverse	samesign	logicalShift	leftBitCount	float_i2f	floatScale2	float64_f2i	floatPower2
36.00	1.00	4.00	4.00	3.00	2.00	3.00	4.00	4.00	4.00	3.00	4.00

test 截图：



解题报告

亮点

1. logtwo
2. leftBitCount
3. byteSwap

题目思路及代码

logtwo

思路

这道题的核心在于寻找整数 `v` 的最高有效位（最高位为1的位）所在的位置。为了实现这一目标，我采用了二分查找的思想，通过移位操作逐步缩小范围，最后确定最高有效位。 具体实现中，有如下几个tips：

1. **移位操作与判断**：本题不能直接使用 `if` 语句判断某个移位后的值是否大于 0，因此我通过 `(v >> n) > 0` 的形式来判断值是否大于 0，并根据结果进行进一步的移位。
2. **二分查找思想**：代码中首先通过右移 16 位来检查高 16 位是否有 1，然后逐步缩小范围（右移 8 位、4 位、2 位等）。每次判断后，如果存在 1，就通过位运算记录当前找到的位位置。

3. **位运算优化**：因为无法使用 `+` 符号，代码通过使用位运算符 `|` 来累加每次找到的位位置，而不是使用加法运算。这是因为每次计算的位移都是 2 的幂次，所以 `|` 运算可以起到加法的作用。

最终，通过不断缩小搜索范围，准确找出最高有效位的位置，返回其对应的数值。

leftBitCount

思路

本题的目标是计算一个数 `x` 的左边连续 1 的个数。解法核心在于通过将 `x` 取反，将问题转化为求最高有效位的位置。这类似于之前的 `logtwo` 问题，但由于本题目中不能使用 `>` 符号进行判断，因此采用了一些特殊处理方式。

具体步骤如下：

1. **取反操作**：首先我们对 `x` 取反，变成 `~x`，这样连续的 1 变成连续的 0，题目变成了计算最高有效 0 的位置。
2. **二分查找的思想**：通过二分法逐步缩小范围，每次右移若干位并检查移位后的结果是否为 0，从而确定当前段是否有 1。为此，使用 `(x >> n) ^ 0` 进行判断，再通过 `!!` 双重非操作来将结果转化为布尔值（1 或 0），并根据结果调整位移量。
3. **调整返回结果**：由于 `x` 取过反，最后结果需要进行一些调整。通过 `33 + ~ans` 来计算 `32 - ans`，最终得出左边连续 1 的个数。

byteSwap

思路

1. **生成掩码并摘取对应字节**：
   - 首先，通过 `n << 3` 和 `m << 3` 计算出字节所在的位偏移量 `a` 和 `b`。一个字节占 8 位，所以这里的位移是以 8 位为单位的。
   - 通过生成字节全为 1 的掩码 `k = 255 << a` 和 `t = 255 << b`，分别得到 `x` 中 `n` 和 `m` 位置字节的值。具体操作为：`x & k` 和 `x & t`，用于将 `x` 的其他位清零，只保留目标字节的数据。
2. **交换字节位置**：
   - 将摘取到的字节右移到低位（通过 `>>` 操作），为了避免符号扩展带来的问题（符号位干扰），使用 `& 0xff` 保证字节的高位清零。
   - 然后将处理后的字节移到目标位置：第 `n` 字节移到第 `m` 字节处，第 `m` 字节移到第 `n` 字节处，使用 `<<` 操作完成左移。
3. **生成最终结果**：
   - 首先，我们将 `x` 中原本 `n` 和 `m` 位置的字节全部置零，方法是 `x & (~(k | t))`，这一步确保要交换的两个字节位置都被清零。
   - 最后，将之前已经交换位置的字节通过 `|` 操作合并到 `x` 中，从而生成最终结果。

bitXor

思路

用 `~(x & y)` 处理两个数中相同为1的位， `~(~x & ~y)` 处理两个数中相同为0的位。

---

samesign

思路

这道题要求判断两个数的符号是否相同。对于整数，符号位是最高位，可以通过右移31位判断符号。

1. **零值的特殊处理**：
   - 0 既不是正数也不是负数，当x和y都为0时，它们被认为具有相同的符号。
2. **符号位的判断**：
   - 对于非零值，取出符号位，比较它们是否相等。

---

reverse

思路

这道题要求将一个32位无符号整数的位顺序反转。我采用了逐位交换的策略，依次交换最低位和最高位，第二低位和第二高位，直到中间。

1. 逐位交换：
   - 每次循环中，使用移位操作分别获取k位和(31-k)位的值，交换它们的位置，并将交换后的结果重新写入原来的位中。
2. 按位操作：
   - 通过构造掩码，将对应的位提取并交换，实现逐位反转。

---

logicalShift

思路

1. 构造掩码：
   - 使用移位操作构造出一个掩码v，该掩码的高n位是0，低位是1。具体实现是通过将 `1 << 31` 右移n位，然后将其反转得到。
2. 逻辑右移：
   - 使用算术右移 `x >> n`，并将其与掩码v按位与操作，保证高位用0填充。

---

float_i2f

代码

```
unsigned float_i2f(int x) {
    if (!x)
        return 0;
    if (x == 0x80000000)
        return 0xCF000000;
    unsigned sign = x & 0x80000000;
    if (sign)
        x = -x;

    int s = 31;
    while (!(x & (1 << s)))
        s--;
    int exp = s + 127;

    unsigned frac = x << (31 - s);
    unsigned extra = frac & 0xFF;
    frac = (frac >> 8) & 0x7FFFFF;
    if ((extra > 128) | ((extra == 128) & (frac & 1))) {
        frac++;
        if (frac == 0x800000) {
            frac = 0;
            exp++;
        }
    }

    return sign | (exp << 23) | frac;
}
```

思路

1. 符号处理：
    - 首先处理x为0的特殊情况。如果x为负数，提取符号位并取负数的绝对值。
2. 指数计算：
    - 通过逐位移位找到x的最高有效位，计算该位对应的指数（即x的二进制表示中的最高位位置减去1，加上偏置值127）。
3. 尾数和舍入处理：
    - 取出x的尾数部分，对尾数进行偶数舍入处理。

---

floatScale2

思路

这道题要求将浮点数放大两倍，即乘以2。

1. 特殊值处理：
    - 如果输入的浮点数是无穷大或NaN，直接返回输入。
2. 非规格化数处理：

- 如果指数部分为0，表示浮点数为非规格化数，此时只需要将尾数左移一位。

3. 规格化数处理：
    - 如果指数部分不为0，将指数加1即可实现乘以2。

---

float64_f2i

思路

1. 指数与溢出判断：
    - 首先根据指数值判断是否发生溢出。如果指数过大，返回溢出值0x80000000；如果指数小于0，则返回0（下溢）。
2. 尾数提取与处理：
    - 将尾数与隐含的1结合，并根据指数的大小进行位移操作，调整尾数的值。
3. 符号处理：
    - 根据符号位决定最终返回的整数是正还是负。

---

floatPower2

思路

这道题要求计算2的x次方的浮点数表示。

1. 指数范围判断：
    - 根据x的大小，判断是否超出浮点数表示的范围。如果x大于127，返回+INF；如果x小于-149，返回0（太小无法表示）。
2. 非规格化数处理：
    - 如果x的范围在-149到-126之间，返回一个非规格化数。
3. 计算结果：
    - 对于正常范围内的指数，直接计算对应的浮点数指数部分即可。

反馈/收获/感悟/总结

这次完成 Datalab 花费了我一个下午加一个晚上的时间，从最初对位运算的陌生，到能自己设计出一些思路来解决问题，我实感觉收获颇丰。

首先，通过这个实验，我对整数和浮点数在机器中的底层表示有了非常细致的理解。特别是在完成 `float_i2f` 这道题目要求我对浮点数的规格化和非规格化表示、以及它们的舍入规则等必须彻底掌握了解。尽管题目的思考过程相当复杂，需要考虑各种边界情况，但不得不承认这是个很有挑战性且富有启发性的好题（但是感觉我的解法很丑陋）。

其次，我也学到了许多位运算的巧妙应用，比如用 `|` 替代加法，用 `~` 替代减法，以及各种掩码操作。这些技巧让我在面对复杂位运算问题时能更灵活高效地解决问题。

总体来说，完成这个 lab 的体验非常不错（赞美助教），但最开始lab发布时感觉bug不少，如果以后能在最初发布时就没有问题就更好了。

参考的重要资料

王晶老师的ppt

浮点数详解（一篇彻底学通浮点数）-CSDN博客