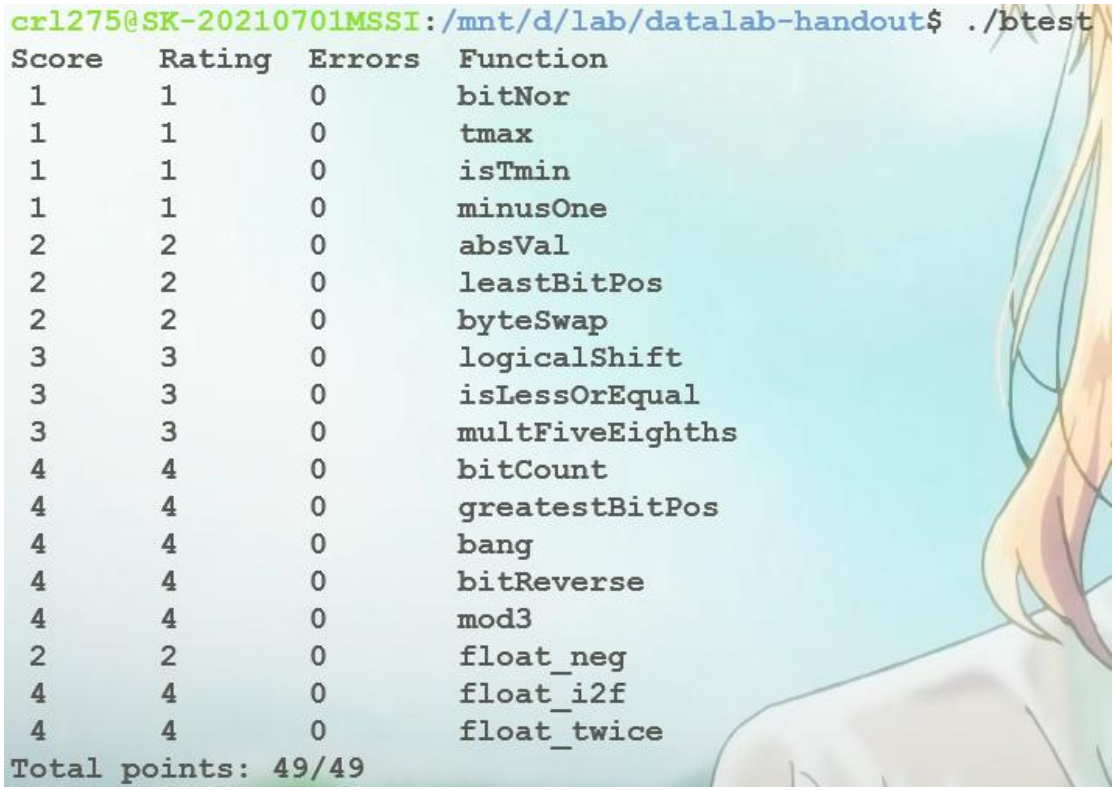
# ICS Lab1-DataLab

姓名：陈锐林 学号：21307130148

一、执行./dlc -e bits.c 的截图：


二、执行./btest 的截图：



三、函数思路：（简单的题写在注释里，难的有拎出来写）

0. 变量：int z（还有一些变量，放在第一次出现的时候；.c 里都在最前面）

```
1. bitNor
int bitNor(int x, int y) {
  z = (~x) & (~y);   //只有当两位都是 0 的时候，结果才为 1
  return z;
}
```

```
2. tmax  //不能直接得到 tmax，就先得其反码，再取反
int tmax(void) {
  z = 1 << 31;
  z = ~z;
  return z;
}
```

```
3.isTmin //Tmin 为 0x80000000，其特点在于与补码异或后为 0
int isTmin(int x) {
  z = x ^ (1 + ~x);
  return !( z | (!x)); //但是 0 也满足这一特点，使 x 为 0 时，也返回 0
}
```

```
4.minusone          //要得到-1，不能用减号，那就将 0 取反
int minusOne(void){
  z = 0;
  return ~z;
}
```

```
5.absVal  //求绝对值，负数要取反加 1，正数不动
int absVal(int x){
  z = x ^ (x >> 31);   //取反或者不取反可以通过异或决定（最高位右移）
  return z + ((x >> 31) & 1); //加不加 1 取决于 x 的最高位
}
```

```
6.leastBitPos  //要 return 这个最低位的值，那么需要把前后都置 0
int leastBitPos(int x) {
  z = x ^ (x + ~0); //构造一个数，让除了最低位都与 x 不同
  return z & x;     //再返回按位与即可
}
```

```
int z1,z2,z3,z4,z5;  //从这开始声明了 5 个变量
7.byteSwap
int byteSwap(int x, int n, int m){
  n = n << 3;
  m = m << 3;
  z1 = 0xff << n;    //为了取出第 n 个字节，之后实现与操作
  z2 = 0xff << m;    //为了取出第 m 个字节，之后实现与操作
  z3 = ((x & z1) >> n) & 0xff; //取出第 n 个字节
  z4 = ((x & z2) >> m) & 0xff; //取出第 m 个字节
  z5 = (z3 << m) | (z4 << n);  //构造只有 m 和 n 字节保存了原数的数
  return (x & ~z1 & ~z2) | z5; //先通过与让 x 撤掉 m/n 字节，再取或
}
```

```
8.logicalShift //逻辑右移，重点在于第一位到底是多少
int logicalShift(int x, int n) {
  z = (~0) + ((1 << (32 + (~n))) << 1); //重点是要把[31 - n + 1,31]
变为 0，~0 即为 0xFFFFFFFF，32 + ~n 即 31 - n,就保证了 z 后面位全为 1
  return ((x >> n) & z); //先移动 x，再将其与 z 取位与即可
}
```

```
9.isLessOrEqual //判断小于等于，先考虑符号，若一正一负，即得结果
int isLessOrEqual(int x, int y) {
  int z1 = ((x & ~y) >> 31) & 1; //z1 为 1 就表征了 x 为负，y 为正
  int z2 = ~((x ^ y) >> 31); //z2 为~1 表征了异号，为~0 表征同号
  int z3 = !(x ^ y);     //考虑一种简单情况，即 x 和 y 相同，即返回 1
  return (z1) | (z3 & 1) | ((z2 & ((x + ~y + 1) >> 31) & 1));
  //z1 为 1 或 z3 为 1 或 同号且 x - y < 0 返回 1，
  //而异号且 x 为正，y 为负时返回 0
}
```

```
10.multFiveEights //首先题目说了 x 是非负数，但先向左移位可能会出错
int multFiveEighths(int x){ //不妨先向右移，再向左移
  z = x & 111;        //后三位需要特殊处理，可能会进位
  z = z + (z << 2); //后三位采取先×5 后÷8
  z = z >> 3;
  x = x & (~111);    //除此之外，x 先÷8 再×5
  x = x >> 3;
  x = x + (x << 2);
  return x + z;      //返回二者的和即可
}
```

11.bitCount

```
int bitCount(int x) {
  z = 0x55 << 8 | 0x55;
  z1 = 0x33 << 8 | 0x33;
  z2 = 0x0f << 8 | 0x0f;
  z3 = 0xff << 16 | 0xff; //z3 为 0x00ff00ff
  z4 = ~0 + (1 << 16);    //z4 为 0x0000ffff
  z |= z << 16;            //z 为 0x55555555
  z1 |=  z1 << 16;         //z1 为 0x33333333
  z2 |=  z2 << 16;         //z2 为 0x0f0f0f0f
  x = (x & z) + ((x >> 1) & z);
  x = (x & z1) + ((x >> 2) & z1);
  x = (x & z2) + ((x >> 4) & z2);
  x = (x & z3) + ((x >> 8) & z3);
  x = (x & z4) + ((x >> 16) & z4);
  return x;
}
```

具体的做法如下，慢慢由两位拓展到四位，再到八位，以此类推。
下面取一个 8 位的数来说明 11010010（可知有 4 个 1）
Step1：与 01010101 取位与，为 01010000；向右移 1 位后再取与为 01000001，相加为 10010001。（从左往右偶数位，如果为 1 会被保存；第二次操作只要奇数位为 1 也会被保存，分成两两一组，即有多少个 1，如 10 01 00 01 即表示有 2/1/0/1 个 1。）
Step2：与 00110011 取位与，为 00010001；向右移 2 位后再取与为 00100000，相加为 00110001。（因为 00110011 的特殊性，会得到（从左往右）第 1、2、5、6 和 3、4、7、8 的 1 的个数，0011 0001 即表示有 3/1 个 1）
Step3：与 00001111 取位与，为 00000001；向右移 4 位后再取与为 00000011，相加为 4。（因为 00001111 的特殊性，只会保留后 4 位的值，那么只要这时候后 4 位表示的是具体的 1 的数目就好啦）

```
12.greatestBitPos
int greatestBitPos(int x) {
  z = 0;
  z += ((!!( x & ((~0) << (z + 16)))) << 4);   (4)
  z += ((!!( x & ((~0) << (z + 8)))) << 3);    (3)
  z += ((!!( x & ((~0) << (z + 4)))) << 2);    (2)
  z += ((!!( x & ((~0) << (z + 2)))) << 1);    (1)
  z += (!!( x & ((~0) << (z + 1))));           (0)
  return (1 << z) & x;
}
```

具体做法：如果每位都试一次会超出字符限制，所以采用二分法。
首先呢判断前（左➡右）16 位是不是有 1，有的话要把 z 加 16;
（1）如果我们假设前 4 次都为 0，那么就很直观得得到第 5 步是在考察倒二位是不是有 1（前面肯定没有了），如果有最后 z = 1，1 << 1 & x，就会得到 2;如果 z 还是为 0，那么直接进入最后一步，考察最后一位有没有 1;
（2）根据代码中的标号，假设在第 i 段，我们得到了 z 要加 2^i 次方的信息，这时候已知前 32 - 2^(i+1)位都是不含 1 的，在第[32 - 2^(i+1) + 1, 32 - 2^(i+1) + 2^i]位是有 1 的，那么接下来的式子，就会考察前 32 - [2^i + 2^(i-1)]位，根据前面已有信息，即考察中间靠左的 2^(i-1)位是否有 1，然后就会把[32 - 2^(i+1) + 1, 32 - 2^(i+1) + 2^i]区间一分为二，靠左的有 1，那就接着看这边；否则就看右边。以次递归，二分得到到底是在哪一位有 1。

```
13.bang  //类似于前面判断是不是 Tmin 那一题，考虑不是 0 的数的特点
int bang(int x) {
  z = (~((~x + 1) | x) >> 31) & 1; //只要不是 0，经过或这一步操作后，会让第一位为 1，根据题目要求，这个时候应该返回 0，所以再取一次反即可
  return z;
}
```

```
14.bitReverse
int bitReverse(int x){
  z = 0x55 << 8 | 0x55;
  z1 = 0x33 << 8 | 0x33;
  z2 = 0x0f << 8 | 0x0f;
  z4 = 0xff << 8 | 0xff;    //z4 = 0x0000ffff
  z3 =  z4 << 8 ^ z4;       //z3 = 0x00ff00ff
  z = z | z << 16;          //z = 0x55555555
  z1 = z1 | z1 << 16;       //z1 = 0x33333333
  z2 = z2 | z2 << 16;       //z2 = 0x0f0f0f0f
  x = ((x & z) << 1)|((x >> 1) & z);
  x = ((x & z1) << 2)|((x >> 2) & z1);
  x = ((x & z2) << 4)|((x >> 4) & z2);
  x = ((x & z3) << 8)|((x >> 8) & z3);
  x = ((x&z4)<<16)|((x>>16)&z4);
  return x;
}
```

有点类似于 bitCount，采用二分法；
采用一个具体的例子 1101 0101 1101 1110 0110 1111 0001 1001
第一行操作后 x 为 1010 1010 1010 1000 1000 1010 0010 0010 与
0100 0000 0100 0101 0001 0101 0000 0100 取或，为
1110 1010 1110 1101 1001 1111 0010 0110 可以观察到即把每两位都取了反；
之后的操作就是每 4 位对称、每 8 位对称、每 16 位对称，即完成饿了 reverese 操作。从理论分析，z 为 0101……0101，可以存下偶数位（左➡右）的 1，x >> 1 再取与，就能存下奇数位的 1，最后取或，即完成了对称，其他行也类似。
但这里每想到如何控制运算符在 40 个，超了 1 个。

```
15.mod3                    //代码很长，纯粹是人太笨
int a,b,c,d,e,f;
int mod3(int x){           //Step1：每两位考虑，利用 2 的奇数次方
  a = (x & 3);             //mod3 余 2，偶数次方余 1，都加起来，最多 48
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 3);
```

```
  x = x >> 2;
  a = a + (x & 3);
  x = x >> 2;
  a = a + (x & 1);
  x = x >> 1;
  a = a + (x & 1);
  b = a & 3;                    //Step2：重复，相加，b 最多为 9
  a = a >> 2;
  b = b + (a & 3);
  a = a >> 2;
  b = b + (a & 3);
  c = (b & 1);                  //从 c 开始改变策略，2 的偶数次幂加 1，
  b = b >> 1;                   //奇数次幂-1
  c = c + (~(b & 1) + 1);
  b = b >> 1;
  c = c + (b & 1);
  b = b >> 1;
  c = c + (~(b & 1) + 1);
  f = !(c | !x);                //为了保证负数 mod3 为-1，-2，0
  d = !(x & 1);                 //设置一些标志，此时 x 为原数最高位
  e = !(c >> 31);               //f/d/e 的设置是根据后面写的完善的
  z = ~e + 1;
  c = c + d + d + d + z + z + z + f + f + f;
  return c;   //对应 6 种情况，原数为负，c 为负、正、零；原数为正，c 为负、正、零。可以发现：原数为负 d 为 0，不加 3；若此时余数为正数，会触发 z 为-1，而导致-3；若余数为 0，会先-3，再由 f 补上，其他情况类似。
}
```

```
16.float_neg
unsigned sign,exp,frac;    //这三个变量定义就像书上规定的三个 parts
unsigned float_neg(unsigned uf) {
  sign = uf >> 31;          //符号
  exp = uf >> 23 & 0xFF;    //阶码
  frac = uf & 0x7FFFFF;     //尾数
  if((exp == 0xFF) && (frac != 0)){
    return uf;
  } //考虑特殊情况，此时为特殊值，NaN
  return uf ^ 0x80000000;   //取反
}
```

```
17.float_i2f
int shift;
unsigned float_i2f(int x) {
  sign = x & 0x80000000;
  shift = 30;                              //初始化 shift
  if(x == 0x80000000) return 0xCF000000; //特殊值,直接表达
  if(!x) return 0;                        //x 为 0 就直接返回了
  if(sign) x = ~x + 1;                   //x 为负，那么先把 x 取反加 1
  while(!(x & (1 << shift))) shift--;   //先找到最高位
  if(shift <= 23){
    x <<= 23 - shift;     //考虑到尾数为 23 位，需补足一下
  }
  else{          //多了，那就要考虑舍去
    x += (1 << (shift - 24)); //why 55? 因为 55-shift 一定小于 32
    if(x << (55 - shift)); else x &= (0xFFFFFFFF << (shift - 22));
    //加了之后，要取舍；如果 x << (55 - shift) 非 0 就保留
    if(x & (1 << shift)); else shift++; //判断进位并且完成尾数构造
    x >>= (shift-23);  //x 最终要减位到 23
  }
  x = x & 0x007FFFFF;        //构造尾数
  shift = (shift + 127) << 23;   //构造阶码,注意 Bias
  return (x) | (shift) | (sign);
}
```

```
18.float_twice
unsigned float_twice(unsigned uf) {
  sign = uf >> 31;
  exp = uf >> 23 & 0xFF;
  frac = uf & 0x7FFFFF;
  if(exp == 0 && frac < 0x7FFFFF) //denormalized values
  frac <<= 1; //虽然是非规格化的值，但是尾数没这么大
  else if(exp == 0 && frac == 0x7FFFFF){ //denormalized values
  exp = 1; //已经触顶了，要让阶码变为1，并且让尾数变小
  frac = 0x7FFFFE;
  }
  else if(exp == 0xFE){//虽然规格化，但马上可能变成特殊值了
  exp = 0xFF;
  frac = 0;
  }
  else if(exp == 0xFF) //special values 特殊值直接返回
  return uf;
  else //normal values 规格化的值阶码加1就好
  exp += 1;
  return (sign << 31) | (exp << 23) | frac;
}
```

## 四、引用与参考：

几题比较不好想的，bitCount、bitReverse、float_i2f(一开始没考虑全)，有参考 csdn 的解答、询问一些在其他学校的高中同学（大佬），然后有经过自己的消化后自己写一遍，注释和解题思路都是自己写的。

## 五、感受与建议：

感受：是对整型和 float 的表示的再次认识，提高挺大，挺长知识。但被虐得很惨……

建议：建议给下一届的搞得更难点，但这学期的能不能简单点。（比如把比较难的搞成选做题或者加分题？）