Lab1 实验报告

一、实验环境

Ubuntu 14.04　64bits

二、实验内容及结果

拿到 Lab 先把要求都读了一遍，看到题目后感觉这不就是智力游戏嘛，做到后来还是输了输了。前面五道位运算的题目相对简单，主要考察位操作：1）利用 Morgan 定律用或非代替与；2）通过移位和与一个常数取到某一位；3）通过算术右移实现逻辑右移，返回一个二进制数中 1 的个数；4）通过取反加 1 返回一个二进制数的补码构成的数；5）寻找 0 和其它数的不同即可。其中比较难的就是 3）返回二进制数中 1 的个数（也是用操作符最多的一道题），采用分治的方法，因为操作数要求不能一位一位检测，可以四位四位检验，然后再相加即可，不太容易想到。

这一部分报错一是开始 make 的时候有“error: sys/ cdefs.h: 没有那个文件或目录”，经查是因为 64 位 ubuntu 系统编译 32 位程序的问题，sudo apt_get install lib32readline-gplv2-dev 即可；二是主要是体现在不认真上：复杂的题目因为考虑得比较详细，写的时候也比较仔细所以倒是会一次过，一些简单的题目会在边界值报错，原因是为了减少操作符的使用，把代码写得比较紧凑，经常会出现少了一个括号或者忘记加括号导致错误的情况。例：

```
qjw@qjw-VirtualBox:~/Lab1/Lab1$ ./btest -f logicalShift
Score   Rating  Errors  Function
ERROR: Test logicalShift(-2147483648[0x80000000],31[0x1f]) fail
ed...
...Gives 0[0x0]. Should be 1[0x1]
Total points: 0/3
```

第二部分是二进制补码运算，主要考察正负数、补码编码：6）32 位带符号数的最小整数即-2147483648=0x80000000；7）判断能否用 n 位表示 fitbits 一开始写法比较啰嗦，

```
/*右移n-1位(>>n+~1)，检验是否所有位都是0，如果是则可以表示
 *分正负数两种情况，正数直接右移，负数取反后右移（最高位是1）
 */
/*int s=x>>31;
return (!s&!(x>>(n+~0)))|(s&!((~x)>>(n+~0)));*/
```

后来经过优化变为：

```
/*如果x是负数能用n位表示，那么y=0
 *如果x是正数能用n位表示，那么y=0xffffffff
 *在这两种情况下，返回1，否则返回0
 */
int y=x>>(n+(~1)+1);
return !y^!(~y);
```

8）除以 2^n 这道题一开始没想到负数的舍入问题，后来加上偏移量成功通过；
9）求相反数只要返回其补码即可；10）判断是否是正数需要注意 0；11）判断是否<=这个题一开始把所有情况都列了出来，后来发现只要分同号和异号即可；12）取对数即找到最高位的 1 所在的位置，也是采用二分法，依次检查高 16 位、高 8 位、高 4 位、高 2 位，最终锁定 1 的位置，类似二分查找（其实如果没有不能使用大数的限制还可以缩小时间复杂度）不过还是不太懂为什么要在开始把所有的变量声明清楚，否则就报错。

第三部分是浮点数的运算，包括返回 float 型相反数；int 型强制转换成 float 型；求 float 型乘 2 的结果。分成三部分符号、阶码和尾数进行即可，需要注意 NaN 和精度。第一遍写漏洞百出（比如精度问题还有少打一位 0），后面慢慢完善（其实是一开始没在 ubuntu 里面做，因为没能登进去 ftp，传不了文件，就先在 VS 里面做，后面直接抄进去了），还有一点就是浮点数的三个题要求里面说可以用大数，这样可以节省操作符数。例：

```
unsigned float_neg(unsigned uf) {
 /*返回一个单精度浮点数（1位符号位+8位阶码+23位尾数）的相反数
  *当x=NaN时，即阶码全为1，尾数不为0，返回NaN
  *当x!=NaN时，直接将符号位取反返回-x即可
  *其中判断8位阶码是否不全为1可用：(uf>>23)^0xff
  *尾数是否部全为0可用：!uf&((1<<23)-1)
  *通过异或0x1<<31改变符号位
  */
 /*判断是否为NaN可以先与0x7fffffff，把符号位变为0，大于0x7f800000则是*/

 /*if((((uf>>23)&0xff)^0xff)||!(uf&((1<<23)-1)))
     return uf^(1<<31);//不为NaN返回-x
   return uf;//否则返回本身*/

 if((uf&0x7fffffff)>0x7f800000)
   return uf;
 return uf^0x80000000;
}

unsigned float_twice(unsigned uf) {
  /*计算uf*2，分三部分处理：符号，阶码和尾数
   *如果阶码部分不为0，将尾数部分左移一位
   *如果阶码部分为0，将阶码+1
   *如果uf是NaN，直接返回NaN，其余数整合三部分后返回
   */
  /*如果uf&0x7f800000==0，说明uf的指数是0，uf是非规格化数/0
   *可以将uf的尾数左移一位实现乘2，通过与0x80000000来保持符号
   *而如果结果不为0，且阶码不是0xff，那么指数直接加1
   *如果阶码为0xff，说明是NaN,直接返回
   */
  /*unsigned s=uf&(1<<31);//取符号位
  unsigned exp=(uf>>23)&0xff;//取阶码
  unsigned frac=uf&((1<<23)-1);//取尾数
  if(exp^0xff){//如果阶码部分不为255，即不为NaN
    if(!exp)//如果阶码不为0
      frac<<=1;//直接将尾数左移一位即可
    else{//如果阶码为0
      exp++;//将阶码加1
      if(exp==0xff)//如果加1后阶码变为255
        frac=0;//将尾数设置为0，表示无穷大
    }
  }
  return s|(exp<<23)|frac;//整合结果*/
  int temp=uf&0x7f800000;
  if(temp==0)
    return ((uf&0x007fffff)<<1|(uf&0x80000000));
  else if(temp!=0x7f800000)
    uf+=0x00800000;
    return uf;
}
```

按要求完善 bits.c 里各个函数，并根据 README 中要求在 Linux 环境下检测是否满足要求，通过 btest 进行测试。

指令如下：

make clean

make btest

./dlc -e bits.c

./driver.pl

实验结果

```
Correctness Results     Perf Results
Points  Rating  Errors  Points  Ops     Puzzle
1       1       0       2       4       bitAnd
2       2       0       2       3       getByte
3       3       0       2       6       logicalShift
4       4       0       2       33      bitCount
4       4       0       2       6       bang
1       1       0       2       1       tmin
2       2       0       2       8       fitsBits
2       2       0       2       7       divpwr2
2       2       0       2       2       negate
3       3       0       2       5       isPositive
3       3       0       2       11      isLessOrEqual
4       4       0       2       27      ilog2
2       2       0       2       3       float_neg
4       4       0       2       29      float_i2f
4       4       0       2       8       float_twice

Score = 71/71 [41/41 Corr + 30/30 Perf] (153 total operators)
```

说明通过了 btest 和 dlc 的检查，最后总共使用了 153 个操作符

三、实验感悟

做完整个实验，感觉收获还是很大的，除了体会到自己很菜之外，学会了一些运算的小技巧，比如说如何取高位/地位/中间某一位，如何判断正负，如何比较大小，如何用其它操作符代替某一操作符等，还有加深了对浮点数的认识，学到了分治的方法。因为有之前做 PA0 的经历，所以这次看英文资料变得简单起来，遇到问题也没有那么慌，自己上网解决或者跟其他同学讨论一般能解决。其实有很多问题还是基础不牢固或者写代码不认真造成的。

第一次接触这种 autograde 的东西，感觉很有意思。最后剩下两个 warning 应该是已有代码本身的问题：

```
qjw@qjw-VirtualBox:~/Lab1/Lab1$ ./dlc -e bits.c
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> includ
ed from includable file /usr/include/stdc-predef.h.
```

```
btest.c: In function ‘main’:
btest.c:528:9: warning: variable ‘errors’ set but not used [-Wunused-but-set-var
iable]
     int errors;
         ^
```