宇佐见函钩 编写

时　　间　　2025 年 4 月 12 日

实验 2：整数与浮点数

int and floats

1. 实验目的

熟悉整型和浮点型数据的编码方式，熟悉 C/C++中的位操作运算。

2. 实验内容

请按照要求补全 bits.c 中的函数，并进行验证。包括以下 7 个函数：理解

No	函数定义	说明

1	`int conditional(int x, int y, int z)`	`/*`
		` * conditional - same as x ? y : z`
		` *   Example: conditional(2,4,5) = 4`
		` *   Legal ops: ! ~ & ^ | + << >>`
		` *   Max ops: 16`
		` *   Rating: 3`
		` */`

2	`int isNonNegative (int x)`	`/* isNonNegative - return 1 if x >= 0,`
		` * return 0 otherwise`
		` * Example: isNonNegative(-1) = 0.`
		` *  isNonNegative(0) = 1.`
		` *   Legal ops: ! ~ & ^ | + << >>`
		` *   Max ops: 6`
		` *   Rating: 3`
		` */`

3	`int isGreater(int x, int y)`	`/* isGreater - if x > y  then return 1,`
		` * else return 0`
		` *   Example: isGreater(4,5) = 0,`
		` *  isGreater(5,4) = 1`
		` *   Legal ops: ! ~ & ^ | + << >>`
		` *   Max ops: 24`
		` *   Rating: 3`
		` */`

4	`int absVal(int x)`	`/*`
		` * absVal - absolute value of x`
		` *   Example: absVal(-1) = 1.`
		` *   You may assume -TMax <= x <= TMax`
		` *   Legal ops: ! ~ & ^ | + << >>`
		` *   Max ops: 10`

No	函数定义	说明
		* Rating: 4 */
5	int isPower2(int x)	/*isPower2 - returns 1 if x is a power of 2, * and 0 otherwise * Examples: isPower2(5) = 0, * isPower2(8) = 1, * isPower2(0) = 0 * Note that no negative number is a power of 2. * Legal ops: ! ~ & ^ | + << >> * Max ops: 20 * Rating: 4 */
6	unsigned float_neg(unsigned uf)	/* * float_neg - Return bit-level equivalent * of expression -f for * floating point argument f. * Both the argument and result are passed * as unsigned int's, but they are to be * interpreted as the bit-level * representations of single-precision * floating point values. * When argument is NaN, return argument. * Legal ops: Any integer/unsigned * operations incl. ||, &&. also if, while * Max ops: 10 * Rating: 2 */
7	unsigned float_i2f(int x)	/* float_i2f - Return bit-level equivalent * of expression (float) x * Result is returned as unsigned int, but * it is to be interpreted as the bit-level * representation of a * single-precision floating point values. * Legal ops: Any integer/unsigned * operations incl. ||, &&. also if, while * Max ops: 30 * Rating: 4 */

3. 实验要求

1）在 Unbuntu18.04LTS 操作系统下，按照实验指导说明书，使用 gcc 工具集编译程序和测试

2）代码符合所给框架代码的规范（详见 bits.c 的开始位置注释内容）

3）需提交：源代码 bits.c、电子版实验报告全文。

4）本实验相关要求：注意：除非函数又特殊说明，违背以下原则均视为程序不正确!!

程序内允许使用：	程序内禁止以下行为：
a. 运算符： ! ~ & ^ | + << >> b. 范围在 0 - 255 之间的常数 c. 局部变量	a. 声明和使用全局变量 b. 声明和使用定义宏 c. 声明和调用其他的函数 d. 类型的强制转换 e. 使用许可范围之外的运算符 f. 使用控制跳转语句：if else switch do while for

4. 实验结果

（可以包括各函数的代码片段、设计思想、执行各测试工具后的结果截图）

```
int conditional(int x, int y, int z) {
  int mask=!!x;
  mask =(mask<<31)>>31;
  return (mask&y)|(~mask&z);
}
```

1、conditional：通过基础运算实现三目运算符的效果。若 x 为真则返回 y，若 x 为假则返回 z。mask 为布尔化的 x。通过左移右移 31 位，将 mask 最低位复制到整个数字。mask 和 y 按位与，~mask 和 z 按位与，通过两者按位或，若 mask 全为 1 则恰好返回 y，若 mask 全为 0 则恰好返回 z

```
int isNonNegative(int x) {
  return !!(~x>>31);
}
```

2、isNonNegative：判断 x 是否为负数，x 取反右移 31 位，x 为正则最低位为 1，x 为负则最低位为 0，返回值转化为布尔类型

```c
int isGreater(int x, int y) {
    int signX = x >> 31 & 1;
    int signY = y >> 31 & 1;
    int signDiff = signX ^ signY;
    int diff = x + (~y + 1);
    int signDiffResult = (diff >> 31) & 1;
    return (!signDiff & !signDiffResult/*1则大于零，0则小于零*/ & !!diff/*diff不为零则是1*/) | (signDiff & signY);
}

/*
```

3、isGreater：核心逻辑分解

Case 1：符号相同时 (signDiff=0)

条件：!signDiff & !signDiffResult & !!diff

!signDiff → 符号相同

!signDiffResult → 差值符号位为 0（差值为正）

!!diff → 差值不为零

意义：当 x 和 y 符号相同且 x 严格大于 y 时返回 1

防溢出原理：符号相同的情况下，x - y 的运算不会溢出（如两个正数相减结果可能为负但不会溢出到符号位错误）

Case 2：符号不同时 (signDiff=1)

条件：signDiff & signY

signY=1 → y 为负数，此时 x 必为正数

意义：当 x 为正数且 y 为负数时直接返回 1（无需计算差值，避免溢出）

关键点：符号不同时通过符号位直接判断，跳过差值计算

示例验证

正数 > 负数

x=5, y=-3 → signDiff=1, signY=1 → 触发 Case 2 → 返回 1

负数 < 正数

x=-2, y=1 → signDiff=1, signY=0 → 不满足任何条件 → 返回 0

相同符号不溢出

x=10, y=5 → signDiff=0, diff=5 → 触发 Case 1 → 返回 1

相同符号潜在溢出

x=INT_MAX, y=0 → diff=INT_MAX → 触发 Case 1 → 返回 1

```c
 */
int absVal(int x) {
  int mask = x >> 31;
  return (x + mask) ^ mask;
}
/*
```

4、absVal：获取 x 的绝对值，创建等于 x 最高位的掩码，

X 为正，Mask=0，(x+0)^0=x,

X 为负，mask=1，(x+1)^1=~(x+1)=-x,得到 x 绝对值。

```
int isPower2(int x) {
  int y=x+~0;
  return !(x&y)&!(x>>31)&(!!x);  //x为2的n次方且x非负且x非零
}
```

5：isPower2：y 为 x-1，若 x 为 2 的 n 次方，则所有位中只有一位包含 1，其余位均为 0，减 1 则得到最高位为 0，更低位均为 1.通过简单运算判断条件，x 是 2 的 n 次方且 x 非负且 x 非零。

```
unsigned float_neg(unsigned uf) {
  unsigned r=uf;
  unsigned t;//符号取反
  r = r^0x80000000 ;
  t = (uf>>23)&0xff;//非数，前八位
  if (t == 0xFF && (uf&0x7FFFFF)>0)//0x7FFFF
  r = uf;
  return r;
}
```

6、float_neg：题目要求将一个无符号类型按照浮点数的形式转化为负数。浮点数的符号位为最高位。

首先翻转符号位（r=r^0x80000000）

然后获取指数位（t=(uf>>23)&0xff）

判断是否为 NaN 的条件：

- 指数字段为 0xFF（全 1）
- 尾数字段非零（0x7FFFFF 为尾数部分掩码）

若是 NaN 则返回原值，若不是 NaN 则返回相反数。

```c
unsigned float_i2f(int x) {
  //即将x变为float型。按照IEEE和向偶取整的规则即可。
  int signbit,highbit,exp,fracbits,flag;
  unsigned temp,result;
  if(!x){
    return x;
  }            //由于规范数情况不包含0，所以先处理0情况。
  signbit=(x>>31)&0x01;
  if(signbit){
    x=~x+1;
  }              //获得符号位，并将x变为正值。
   highbit=0;
  temp=x;
  while(!(temp&0x80000000)){
     temp<<=1;
     highbit++;
  }                    //获得x的最高有效位，即确定fraction的位数。
  temp=temp<<1;
  exp=127+31-highbit;  //根据单精度浮点数规则计算出exp，和fraction位数。
  fracbits=31-highbit;
  flag=0;               //进行向偶舍入
  if((temp&0x1ff)>0x100){
    flag=1;
  }             //出现在规定位置后大于0b100的情况就进1.
  if((temp&0x3ff)==0x300){
    flag=1;
  }          //出现最后一位为1，且下一位为1的情况也进1(向偶取整)
  result=(signbit<<31)+(exp<<23)+(temp>>9)+flag;//计算最终结果
  return result;
}
```

7、float_i2f，下图为代码注释。实验要求将整形数据转化为浮点型数据

```c
unsigned float_i2f(int x) {
    int signbit, highbit, exp, fracbits, flag;
    unsigned temp, result;

    // 处理输入为0的特殊情况
    if (!x) {
        return x; // 浮点数0的表示为全0（符号、指数、尾数均为0）
    }

    // 符号位提取与数值转换
    signbit = (x >> 31) & 0x01; // 提取符号位（0为正，1为负）
    if (signbit) {
        x = ~x + 1; // 负数转正数（补码操作）
    }

    // 定位最高有效位（隐含的leading 1）
    highbit = 0;
    temp = x;
    while (!(temp & 0x80000000)) { // 寻找最高位为1的位置
        temp <<= 1;
        highbit++;
    }
    temp <<= 1; // 左移1位消除leading 1（浮点数尾数隐含最高位1）

    // 计算指数和尾数位数
    exp = 127 + 31 - highbit; // 指数偏移值计算（IEEE 754标准）
    fracbits = 31 - highbit;  // 有效尾数位数（不含leading 1）

    // 向偶舍入处理
    flag = 0;
    // Case 1: 舍入部分超过0.5时进位
    if ((temp & 0x1ff) > 0x100) { // 判断第9-1位是否大于0x100（二进制0.5）
        flag = 1;
    }
    // Case 2: 舍入部分等于0.5且当前尾数末位为1时进位（使结果为偶数）
    if ((temp & 0x3ff) == 0x300) { // 检查第10-1位是否为0x300（二进制0.11）
        flag = 1;
    }

    // 组合最终结果
    result = (signbit << 31) | (exp << 23) | ((temp >> 9) + flag);
    return result;
}
```

5. 实验总结及心得体会

有了实验一的经验总结，本次实验我能够更熟练的运用掩码，移位运算等运算方式，部分题目查询了 csdn 等开源网站并对不懂的知识进行了补充。

实验重点：整形的操作，整形与浮点数的转换，浮点数的操作。

浮点数由符号位（S），尾数（M）和阶码（E）三部分组成，在内存中由 s，exp 和 frac 三部分编码。Exp=E+bias.

浮点数有两种特殊情况需要考虑，分别是 exp=000…0 时和 exp=111…1 时。exp 全为 0 时 E=-bias+1,浮点数表示的是比 1 小的小数。当 exp=111…1 且 frac=000…0，表示正无穷，当 exp=111…1 且 frac≠000…0，浮点数表示无法确定，NaN。这些情况都是需要在实验中考虑到的。