天津大学

计算机系统基础上机实验报告

实验题目 2：整数与浮点数 int and floats

学院名称 智能与计算学部

专　　业 工科试验班

学生姓名 张明君

学　　号 6319000359

年　　级 2019 级

班　　级 留学生班

时　　间 2020 年 6 月 20 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

INTEGER CODING RULES:

Replace the "return" statement in each function with one
or more lines of C code that implements the function. Your code
must conform to the following style:

```
int Funct(arg1, arg2, ...) {
    /* brief description of how your implementation works */
    int var1 = Expr1;
    ...
    int varM = ExprM;

    varJ = ExprJ;
    ...
    varN = ExprN;
    return ExprR;
}
```

Each "Expr" is an expression using ONLY the following:
1. Integer constants 0 through 255 (0xFF), inclusive. You are
    not allowed to use big constants such as 0xffffffff.
2. Function arguments and local variables (no global variables).
3. Unary integer operations ! ~
4. Binary integer operations & ^ | + << >>

Some of the problems restrict the set of allowed operators even further. Each "Expr" may consist of multiple operators. You are not restricted to one operator per line.

You are expressly forbidden to:

1. Use any control constructs such as if, do, while, for, switch, etc.
2. Define or use any macros.
3. Define any additional functions in this file.
4. Call any functions.
5. Use any other operations, such as &&, ||, -, or ?:
6. Use any form of casting.
7. Use any data type other than int. This implies that you cannot use arrays, structs, or unions.

You may assume that your machine:

1. Uses 2s complement, 32-bit representations of integers.
2. Performs right shifts arithmetically.
3. Has unpredictable behavior when shifting an integer by more than the word size.

FLOATING POINT CODING RULES

For the problems that require you to implent floating-point operations, the coding rules are less strict. You are allowed to use looping and conditional control. You are allowed to use both ints and unsigneds. You can use arbitrary integer and unsigned constants.

You are expressly forbidden to:

1. Define or use any macros.
2. Define any additional functions in this file.
3. Call any functions.
4. Use any form of casting.
5. Use any data type other than int or unsigned. This means that you cannot use arrays, structs, or unions.
6. Use any floating point data types, operations, or constants.

NOTES:

1. Use the dlc (data lab checker) compiler (described in the handout) to check the legality of your solutions.
2. Each function has a maximum number of operators (! ~ & ^ | + << >>) that you are allowed to use for your implementation of the function.

The max operator count is checked by dlc. Note that '=' is not counted; you may use as many of these as you want without penalty.

3. Use the btest test harness to check your functions for correctness.
4. Use the BDD checker to formally verify your functions
5. The maximum number of ops for each function is given in the header comment for each function. If there are any inconsistencies between the maximum ops in the writeup and in this file, consider this file the authoritative source.

```
/*
 * STEP 2: Modify the following functions according the coding rules.
 *
 *   IMPORTANT. TO AVOID GRADING SURPRISES:
 *   1. Use the dlc compiler to check that your solutions conform
 *      to the coding rules.
 *   2. Use the BDD checker to formally verify that your solutions produce
 *      the correct answers.
 */


#endif
```

1.conditional

```
/*
 * conditional - same as x ? y : z
 *   Example: conditional(2,4,5) = 4
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 16
 *   Rating: 3
 */
int conditional(int x, int y, int z) {
  x = (!!x)<<31>>31;

  return (y&x)|(z&~x);
}
```

```
If the value of X is 0x00000000 or 0xFFFFFFFF. The answer is (X &amp;amp; Y) |
(~ x &amp;amp; z).
If x! =When 0, x = 0xFFFFFFFF. x = (!!x)<<31>>31
```

2.isNonNegative

```
/*
 * isNonNegative - return 1 if x >= 0, return 0 otherwise
 *      Example: isNonNegative(-1) = 0.    isNonNegative(0) = 1.
 *      Legal ops: ! ~ & ^ | + << >>
 *      Max ops: 6
 *      Rating: 3
 */
int isNonNegative(int x) {
   return (~(x>>31)&1);
}
```

```
Judge the highest position. if 1 Return 1, otherwise.
```

3.isGreater

```
/*
 * 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
 */
int isGreater(int x, int y) {
   int sign_ = ((~x&y)>>31)&1;
   int mark_ = ~((x^y)>>31);
   int equl_ = !!(x^y);
   return sign_ | ((mark_)&(~(x+~y+1))>>31&equl_);
}
```

```
The above code will return to 1. Mark when x > 0, y < 0, or when x and Y symbols

are the same_ =~ ((x ^ y) > > 31), when x + ~ y + 1 > 0, X! = y, equl_ = !!(x^y)=1,

x+~y+1>=0, (~(x+~y+1))>>31 = 0xffffffff
```

4.absVal

```
/*
 * absVal - absolute value of x
 *      Example: absVal(-1) = 1.
 *      You may assume -TMax <= x <= TMax
```

```
 *      Legal ops: ! ~ & ^ | + << >>
 *      Max ops: 10
 *      Rating: 4
 */
int absVal(int x) {
   return (x^(x>>31))+((x>>31)&1);
}
```

```
|x|={-x (~x+1) ,x<0x,x⩾0

X processing can be divided into two cases, taking the inverse + 1 and the
invariant + 0.
As we all know, a number can be exclusive or 1 if it is inversed and exclusive
or 0 if it is invariant.
When x < 0, x > > 31 is 0xFFFFFF, and x ^ (x > > 31) is the reverse, (x > > 31)
&amp; 1 is 0x 1.
```

5.isPower2

```
/*
 * 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
 */
int isPower2(int x) {
   int ret = ((!(x&(x+~0))) & ((~(x>>31)&(!!x))));
   return ret;
}
```

```
The problem is that there can only be one number with bit 1 in binary number
X. Also exclude negative and positive numbers. We find that if x = = 2 ^ n, then
the characteristic of X-1 is that [n-1,0] bits are all 1, so x &amp; (x-1) =
0. Using this property and excluding special cases, we can get the answer.
```

6.float_neg

```
/*
 * 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
 */
unsigned float_neg(unsigned uf) {
   if((uf&0x7fffffff) > 0x7f800000)
      return uf;
  return uf ^ 0x80000000;
}
```

```
It can be judged directly by UF = = UG, but we need to pay attention to + 0
/ - 0 and Nan.
When judging Nan, the index segment is 0xff, the decimal segment is not all 0,
(UF &amp;amp; 0x7fffffff) > 0x7f800000.
```

7.float_i2f

```
/*
 * 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
 */
unsigned float_i2f(int x) {
   int s_ = x&0x80000000;
   int n_ = 30;
   if(!x) return 0;
   if(x==0x80000000) return 0xcf000000;
   if(s_) x = ~x+1;
   while(!(x&(1<<n_))) n_--;
   if(n_<=23) x<<=(23-n_);

   else{
      x+=(1<<(n_-24));
      if(x<<(55-n_)) ;else x&=(0xffffffff<<(n_-22));
      if(x&(1<<n_))    ;else n_++;
      x >>= (n_-23);
   }
   x=x&0x007fffff;
```

```
    n_=(n_+127)<<23;
    return x|n_|s_;
}
```

Same as the previous question, it is divided into three parts to obtain the symbol bit s_ =If x-0x80000000 is a negative number - X and becomes a positive number, then 0x80000000 is treated separately as a special case. Considering the special case, 0x 0 and 0x80000000 are directly returned as 0 and 0xcf000000.

Get the location of the highest 1, while (! (X & (1 < < n_))) n_ --;。

If n_ The number of < = 23 needs to be moved to the left to the beginning of the decimal part (place 1 on the 23rd place), if (n_ <=23) x<<=(23-n_);。

If n_ >23 this number needs to be moved to the right to the starting position of the decimal part (place 1 at the 23rd place). At this time, the rounding problem of the moving part needs to be considered. If the moving part is greater than 0.5, it will be rounded up; if it is less than 0.5, it will be rounded down; if it is equal to 0.5, it will be rounded up.

First, consider the case of > = 0.5 equally, and round up x + = (1 < (n_ -24))。

If = = 0.5, if the rounding condition is odd, we need to change the - 1 operation to even, that is, the lowest 1 to 0, X & = (0xFFFFFF < (n_ -22)), if rounding up produces carry in the highest position, it is necessary to add carry if (X & (1 < < n_)) ;else n_ ++;。 Then you can splice floating-point numbers.