

CSAPP 2019 春季学期

Data Lab: Manipulating Bits

1. 简介

这次作业你要解开15道编程谜题，以熟悉整型及浮点数的位级表示。

2. 操作步骤

从群共享中下载 datalab-handout.tar 文件后，拷贝到你的 Linux 机器（虚拟机）上，输入命令：

```
linux> tar xvf datalab-handout.tar
```

解压缩后会得到多个文件，其中**只需要修改 bits.c 文件**。

bits.c 文件包含 15 个编程谜题的框架。你的任务是完成每个函数的框架，只能使用 straightline 的代码（也就是，不能有循环或者条件语句），也只能使用有限数量的 C 语言算术和逻辑运算符。具体来说就是，你只能使用下述八种运算符：

! ~ & ^ | + << >>

有一些函数有更严格的要求。同时，你也不能使用超过 8 位的常数。bits.c 的注释中有详细的解释，以及我们所期望的代码风格。

另外，还需要从课代表处获得以你学号命名的一个包含一个字符串的文件，将该文件重命名为“stuID”，将重命名后的文件拷贝到 datalab-handout 目录下。否则你将无法正确提交你的成绩。

3. 谜题

这部分描述bits.c中待实现的15个函数功能，一共分为三种类型。

表 1 位运算

函数名称	功能描述	难度等级	最多操作数
bitAnd(x, y)	只用~和 实现 x&y	1	14
getByte(x, n)	从字x中取出第n个字节	2	6
logicalShift(x, n)	逻辑右移	3	20
bitCount(x)	计算x中1的数目	4	40
bang(x)	计算!x，但是不允许使用!	4	12

表 2 二进制补码运算

函数名称	功能描述	难度等级	最多操作数
tmin()	返回最小的补码	1	4
fitsBits(x,n)	x的补码是否可以表示成n位	2	15
divpwr2(x,n)	计算x/2 ⁿ	2	15
negate(x)	不用负号得到-x	2	5
isPositive(x)	如果x大于0返回1	3	8
isLessOrEqual(x,y)	x <= y?	3	24
ilog2(x)	计算⌊log ₂ (x)⌋（向下取整）	4	90

表 3 浮点数运算

函数名称	功能描述	难度等级	最多操作数
float_neg(uf)	计算-f的位级表达	2	10

float_i2f(x)	计算(float)x	4	30
float_twice(uf)	计算2*f	4	30

bits.c文件:

这个是源码文件，里面包含了 15 个待实现的函数，已经给出函数原型。实验内容是按照每个函数的要求编写实现其功能的代码。例如：

```

/*
 * bitXor - x^y using only ~ and &
 *   Example: bitXor(4, 5) = 1
 *   Legal ops: ~ &
 *   Max ops: 14
 *   Rating: 1
 */
int bitXor(int x, int y) {
    return 2;
}

```

函数名: bitXor 参数:int, int

功能: 实现 x^y

要求: 只能使用 \sim 和 $\&$ 运算符, 将结果返回。

如例子所示, 你需要做的就是将 bits.c 文件中的每个函数都按照要求实现, 总体来说就是使用有限的操作实现要求的功能, 上面的例子就是使用两个运算符 \sim 和 $\&$ 来实现 \wedge 运算符的功能, 并且运算符的个数不能超过 Max ops:14 个, 这就需要你先去推理如何用 \sim 和 $\&$ 实现 \wedge , 然后写出表达式。

请同学们务必认真阅读 bits.c 文件中的说明、注意事项和示例。

4. 实验操作

第一步: 将 datalab-handout.tar 文件上传到一台 Linux 机器上, 执行如下命令解压:

```
linux> tar xvf datalab-handout.tar
```

你将看到一个名为 datalab-handout 的目录。

第二步: 实现 bits.c 中的函数, 使用 dlc 编译器检查代码是否满足编码要求, 命令如下:

```
linux> ./dlc bits.c
```

如果没有问题, 则不返回任何提示。

第三步: 使用 btest 程序测试函数功能正确性。编译 btest 程序并进行测试, 命令如下:

```
linux> make btest
```

```
linux> ./btest
```

注意, 只要修改 bits.c 文件, 就需要重新编译 btest 程序, 命令如下:

```
linux> make clean
```

```
linux> make btest
```

btest 程序将自动运行很多组测试用例来检查你实现的函数, 下面是一些 btest 的使用技巧:

```
linux> ./btest -h #输出 btest 命令的帮助信息
```

```
linux> ./btest -f foo #测试指定函数 foo 的正确性
```

```
linux> ./btest -f foo -1 27 -2 0xf #指定输入参数, 测试函数 foo 的正确性
```

注意：如果对实验操作有不懂的地方，可自行阅读 `datalab-handout` 目录中的 `README` 文件。

5. 提交要求

在目录下输入

```
linux> ./driver.pl -u “你的学号”
```

可以在 <http://54.187.73.103:8900/>下直接看到你提交的作业成绩。

结果提交截止时间：**2019 年 4 月 18 日 24:00**

6. 注意事项

- (1) 如果编译时发现很多头文件找不到，尝试执行如下命令安装必要的库：

```
sudo apt-get install build-essential libc6-dev libc6-dev-i386
```

```
sudo apt-get install gcc-4.7-multilib g++-4.7-multilib
```

- (2) 如果你的机器是 32 位的，那么将 Makefile 中

```
CFLAGS = -O -Wall -m64 -std=gnu89
```

改为 `CFLAGS = -O -Wall -m32 -std=gnu89`

7. `bits.c` 中的要求

```
/*
```

```
* Instructions to Students:
```

```
*
```

```
* STEP 1: Read the following instructions carefully.
```

```
*/
```

You will provide your solution to the Data Lab by editing the collection of functions in this source file.

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.

EXAMPLES OF ACCEPTABLE CODING STYLE:

```
/*
 * pow2plus1 - returns 2^x + 1, where 0 <= x <= 31
 */
int pow2plus1(int x) {
    /* exploit ability of shifts to compute powers of 2 */
    return (1 << x) + 1;
}
```

```
/*
 * pow2plus4 - returns 2^x + 4, where 0 <= x <= 31
 */
int pow2plus4(int x) {
    /* exploit ability of shifts to compute powers of 2 */
```

```

int result = (1 << x);
result += 4;
return result;
}

```

FLOATING POINT CODING RULES

For the problems that require you to implement 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.

* /