Data Lab: Manipulating Bits

Introduction

The purpose of this assignment is to become more familiar with bit-level representations and manipulations. You'll do this by solving a series of programming ``puzzles." Many of these puzzles are quite artificial, but you'll find yourself thinking much more about bits in working your way through them and you will learn a lot about how computers represent integer data.

Getting Started

Unzip `dlab-handout.zip` to a directory in which you plan to do your work. This will cause 12 files to be unpacked into the directory:

`bits.c`	The data puzzles that you will solve. **This is the file you will be modifying and handing in.**
`dlab.dsw` `dlab.dsp`	Project and workspace files.
`README`	Helpful information about the lab

`btest.c`
`btest.h`
`decl.c`
`test.c`
`getopt.c`
`bits.h`
`getopt.h`
`tailor.h`
	The test driver and its helper files

Open the `dlab.dsw` workspace from VC++ and you are ready to begin solving your puzzles.

Your Task

The only file you will be modifying and turning in is `bits.c`. Looking at `bits.c` you'll notice a C structure called `info` into which you should insert your name and login ID. Do this right away so you don't forget, as the driver will not run without it.

The bits. c file also contains a skeleton for each of the 10 programming puzzles. Your assignment is to complete each function skeleton using only *straightline* code (i.e., no loops or conditionals) and a limited number of C arithmetic and logical operators. Specifically, you are *only* allowed to use the following eight operators:

```
 ! ~ & ^ | + << >>
```

A few of the functions further restrict this list. See the comments in bits. c for detailed rules and a discussion of the desired coding style.

The enclosed dlab. dsw workspace will help you compile your bits. c file, along with the other helper functions, and link them all together to form the executable driver program btest. exe. The btest. exe driver program allows you to evaluate the functional correctness of your code. Every time you modify one of the puzzles in bits. c, you can check its correctness by rebuilding and rerunning btest. exe.

Puzzles

The following table describes the 10 puzzles that you will be solving in bits. c. The ``Rating" field gives the difficulty rating (the number of points) for the puzzle, and the ``Max ops" field gives the maximum number of operators you are allowed to use to implement each function.

Name	Description	Rating	Max Ops
bitAnd(x, y)	Returns (x & y) using only ~ and |	1	8
bitOr(x, y)	Returns (x | y) using only ~ and &	1	8
isZero(x)	Returns 1 if x == 0 and 0 otherwise	1	2
minusOne()	Returns a value of -1	1	2
tmax()	Returns the maximum two's complement integer	1	4
bitXor(x, y)	Returns (x ^ y) using only ~ and &	2	14
getByte(x)	Extract byte number n from word x	2	14
isEqual(x, y)	Returns 1 if x == y, and 0 otherwise	2	5
negate(x)	Returns -x	2	5
isPositive(x)	Returns 1 if x > 0, and 0 otherwise	3	8

- Function bitAnd computes the And function. That is, when applied to arguments x and y, it returns (x & y). You may only use the operators ~ and |.
- Similarly, function bitOr computes the Or function. That is, when applied to arguments x and y, it returns (x | y). You may only use the operators ~ and &.
- A *predicate* is a function that returns either true or false. Function isZero is a predicate that returns a value of 1 (true) if x is equal to zero. Otherwise, it returns a value of 0 (false).
- Function minusOne takes no arguments and always returns a value of -1, without using the minus operator.
- Function tmax, which also takes no arguments, returns the largest positive two's complement number.

- Function bitXor should duplicate the behavior of the bitwise Xor operation ^, using only the operations & and ~.
- Function getByte extracts a byte from a word and returns that byte. The bytes within a word are ordered from 0 (least significant) to 3 (most significant). For example, getByte(0x12345678, 1) = 0x56
- Function isEqual compares x to y for equality, returning 1 (true) if x == y and 0 (false) otherwise.
- Function negate computes and returns the negative of its input argument x, without using the minus operator.
- Function isPositive is a predicate that returns 1 (true) if its input argument is greater than zero. Otherwise, it returns 0 (false).

Evaluation

Your score will be computed out of a maximum of 30 points based on the following distribution:

16 Correctness of code as reported by btest. exe
(no credit for a puzzle if your instructor determines that you have used an illegal operator).
10 Performance of code, based on number of operators used in each function (maximum of 1 point per puzzle).
4 Style points, based on your instructor's subjective evaluation of the quality of your solutions and your comments.

The 10 puzzles you must solve have been given a difficulty rating between 1 and 3, such that their weighted sum totals to 16. Your instructor will evaluate your functions using the same btest. exe driver that you are using. You will get full credit for a puzzle if it passes all of the tests performed by btest. exe, half credit if it fails one test, and no credit otherwise. You receive no credit if you use an illegal operator for your solution, so pay close attention to the list of allowed operators for each puzzle in bits. c.

Regarding performance, our main concern at this point in the course is that you can get the right answer. However, we want to instill in you a sense of keeping things as short and simple as you can. Thus, for each function we've established a maximum number of operators that you are allowed to use for each function. This limit is very generous and is designed only to catch egregiously inefficient solutions. You will receive one point for each function that satisfies the operator limit.

Finally, we've reserved four points for a subjective evaluation of the style of your solutions and your commenting. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

Advice

- Read the file README for information on running the btest. exe program.

- You'll find it helpful to work through the functions one at a time, testing each one as you go. You can use the `-f` flag to instruct `btest.exe` to test only a single function. For example, "`btest.exe -f bitAnd`".
- The "`-g`" flag is very handy for producing a concise summary of your correctness results. For example, "`btest -g`".
- `btest.exe` is a console application, so you'll need to run it from the Windows command line. On a Windows 2000 system, you can find this at "Start->Accessories->Command Prompt". If you try to run it using "Start->Run.." instead, you won't see any of the output.

Hand In Instructions

Before submitting your solution:

- Make sure you have included your identifying information in your file `bits.c`.

Remove any extraneous print statements that you might have included for debugging.