
The CS:APP Data Lab
Directions to Students

Your goal is to modify your copy of bits.c so that it passes all the
tests in btest without violating any of the coding guidelines.

---

0. Files:

Makefile - Makes btest, fshow, and ishow
README - This file
bits.c - The file you will be modifying and handing in
bits.h - Header file
btest.c - The main btest program
btest.h - Used to build btest
decl.c - Used to build btest
tests.c - Used to build btest
tests-header.c- Used to build btest
dlc* - Rule checking compiler binary (data lab compiler)
driver.pl* - Driver program that uses btest and dlc to autograde bits.c
Driverhdrs.pm - Header file for optional "Beat the Prof" contest
fshow.c - Utility for examining floating-point representations
ishow.c - Utility for examining integer representations

---

1. Modifying bits.c and checking it for compliance with dlc

IMPORTANT: Carefully read the instructions in the bits.c file before
you start. These give the coding rules that you will need to follow if
you want full credit.

Use the dlc compiler (./dlc) to automatically check your version of
bits.c for compliance with the coding guidelines:

```
unix> ./dlc bits.c
```

dlc returns silently if there are no problems with your code.
Otherwise it prints messages that flag any problems. Running dlc with
the -e switch:

```
unix> ./dlc -e bits.c
```

causes dlc to print counts of the number of operators used by each function.

Once you have a legal solution, you can test it for correctness using
the ./btest program.

---

2. Testing with btest

The Makefile in this directory compiles your version of bits.c with
additional code to create a program (or test harness) named btest.

To compile and run the btest program, type:

```
unix> make btest
unix> ./btest [optional cmd line args]
```

You will need to recompile btest each time you change your bits.c
program. When moving from one platform to another, you will want to
get rid of the old version of btest and generate a new one. Use the
commands:

```
unix> make clean
unix> make btest
```

Btest tests your code for correctness by running millions of test
cases on each function. It tests wide swaths around well known corner
cases such as Tmin and zero for integer puzzles, and zero, inf, and
the boundary between denormalized and normalized numbers for floating
point puzzles. When btest detects an error in one of your functions,
it prints out the test that failed, the incorrect result, and the
expected result, and then terminates the testing for that function.

Here are the command line options for btest:

unix> ./btest -h
Usage: ./btest [-hg] [-r <n>] [-f <name> [-1|-2|-3 <val>]*] [-T <time limit>]

-1 <val> Specify first function argument
-2 <val> Specify second function argument
-3 <val> Specify third function argument
-f <name> Test only the named function
-g Format output for autograding with no error messages
-h Print this message
-r <n> Give uniform weight of n for all problems
-T <lim> Set timeout limit to lim

Examples:

Test all functions for correctness and print out error messages:
unix> ./btest

Test all functions in a compact form with no error messages:
unix> ./btest -g

Test function foo for correctness:
unix> ./btest -f foo

Test function foo for correctness with specific arguments:
unix> ./btest -f foo -1 27 -2 0xf

Btest does not check your code for compliance with the coding guidelines. Use dlc to do that.

---

3. Helper Programs

We have included the ishow and fshow programs to help you decipher integer and floating point representations respectively. Each takes a single decimal or hex number as an argument. To build them type:

```
unix> make
```

Example usages:

```
unix> ./ishow 0x27
Hex = 0x00000027,       Signed = 39,    Unsigned = 39

unix> ./ishow 27
Hex = 0x0000001b,       Signed = 27,    Unsigned = 27

unix> ./fshow 0x15213243
Floating point value 3.255334057e-26
Bit Representation 0x15213243, sign = 0, exponent = 0x2a, fraction = 0x213243
```

```
Normalized.  +1.2593463659 X 2^(-85)

linux> ./fshow 15213243
Floating point value 2.131829405e-38
Bit Representation 0x00e822bb, sign = 0, exponent = 0x01, fraction = 0x6822bb
Normalized.  +1.8135598898 X 2^(-126)
```