```
##############################################################################
# CS:APP Data Lab
# Directions to Instructors
#
# May 31, 2011: Now includes the "Beat the Prof" contest
#
# Copyright (c) 2002-2018, R. Bryant and D. O'Hallaron
##############################################################################
#

This directory contains the files that you will need to run the
CS:APP
Data Lab, which helps develops the student's understanding of bit
representations, two's complement arithmetic, and IEEE floating
point.

For fun, we've also provided a new user-level HTTP-based "Beat the
Prof" contest that replaces the old email-based version. The new
contest is completely self-contained and does not require root
password.  The only requirement is that you have a user account on a
Linux machine with an IP address.

System requirements: Uses bison and flex to build dlc.

************
1. Overview
************

In this lab, students work on a C file, called bits.c, that consists
of a series of programming "puzzles".  Each puzzle is an empty
function body that must be completed so that it implements a
specified
mathematical function, such as "absolute value". Students must solve
the non-floating point puzzles using only straight-line C code and a
restricted set of C arithmetic and logical operators. For the
floating-point puzzles they can use conditionals and arbitrary
operators.

Students use the following three tools to check their work.
Instructors use the same tools to assign grades.

1. dlc: A "data lab compiler" that checks each function in bits.c for
compliance with the coding guidelines, checking that the students use
less than the maximum number of operators, that they use only
straight-line code, and that they use only legal operators. The
sources and a Linux binary are included with the lab.

2. btest: A test harness that checks the functions in bits.c for
correctness. This tool has been significantly improved, now checking
wide swaths around the edge cases for integers and floating point
representations such as 0, Tmin, denorm-norm boundary, and inf.

3. driver.pl: A autograding driver program that uses dlc and btest to
check each test function in bits.c for correctness and adherence to
the coding guidelines

The default version of the lab consists of 15 puzzles, in
./src/selections.c, chosen from a set of 73 standard puzzles defined
in the directory ./src/puzzles/. You can customize the lab from term
to term by choosing a different set of puzzles from the standard set
of puzzles.

You can also define new puzzles of your own and add them to the
standard set. See ./src/README for instructions on how to add new
puzzles to the standard set.

NOTE: If you define new puzzles, please send them to me (Dave
O'Hallaron, droh@cs.cmu.edu) so that I can add them to the standard
set of puzzles in the data lab distribution.

********
2. Files
********
```

All CS:APP labs have the same simple top-level directory structure:

```
Makefile           Builds the entire lab.
README             This file.
src/               Contains all source files for the lab.
datalab-handout/   Handout directory that goes to the students.
Generated
                   by the Makefile from files in ./src. Never modify
anything
                   in this directory.
grade/             Autograding scripts that instructors can use to
                   grade student handins.
writeup/           Sample Latex lab writeup.
contest/           Everything needed for the optional "Beat the Prof"
contest.
```

**********
3. Building the Lab
**********

Step 0. If you decide to run the "Beat the Prof" contest (section 5), then edit the ./contest/Contest.pm file so that the driver knows where
to send the results. See ./contest/README for the simple instructions. If you decide *not* to offer the contest, then do nothing in this step.

Step 1. Select the puzzles you want to include by editing the file ./src/selections.c.

The default ./src/selections.c comes from a previous instance of the Data Lab at CMU. The file ./src/selections-all.c contains the complete list of puzzles to choose from.

Step 2. Modify the Latex lab writeup in ./writeup/datalab.tex to tailor it for your course.

Step 3. Type the following in the current directory:

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

The Makefile generates the btest source files, builds the dlc binary (if necessary), formats the lab writeup, and then copies btest, the dlc binary, and the driver to the handout directory. After that, it builds a tarfile of the handout directory (in ./datalab-handout.tar) that you can then hand out to students.

Note on Binary Portability: dlc is distributed in the datalab-handout directory as a binary. Linux binaries are not always portable across distributions due to different versions of dynamic libraries. You'll need to be careful to compile dlc on a machine that is compatible with
those that the students will be using.

Note: Running "make" also automatically generates the solutions to the
puzzles, which you can find in ./src/bits.c and ./src/bits.c-solution.

**********
4. Grading the Lab
**********

There is a handy autograder script that automatically grades your students' handins. See ./grade/README for instructions.

**********
5. "Beat the Prof" Contest
**********

For fun, we've included an optional "Beat the Prof" contest, where students compete against themselves and the instructor. The goal is to

solve each Data Lab puzzle using the fewest number of
operators. Students who match or beat the instructor's operator count
for each puzzle are winners. See ./contest/README for the simple
instructions on how to set up the contest.

NOTE: The contest is completely optional. Whether you decide to
offer it or not has no affect on how you build and distribute the
lab.

NOTE: If you do decide to offer the contest, then you should
configure
the contest *before* you build the lab, so that the driver knows the
server and port to send the contest results of each student (using
the
constants defined in the ./src/Driverhdrs.pm file, which is
autogenerated from the ./contest/Contest.pm config file).

If you decide to offer the contest *after* you've built and handed
out
the lab to the students, you're still OK:

1) Configure the contest as described in contest/Makefile

2) Rebuild the lab in the usual way:
   linux> cd datalab
   linux> make

3) Distribute the new ./src/Driverhdrs.pm file to the students.

**********
6. Experimental BDD checker
**********

For fun, we have included an experimental correctness checker based
on
binary decision diagrams (BDDs) (R. E. Bryant, IEEE Transactions on
Computers, August, 1986) that uses the CUDD BDD package from the
University of Colorado. The BDD checker does an exhaustive test of
the
test functions in bits.c, formally verifying the correctness of each
test function against a reference solution for *ALL* possible input
values. For functions that differ from the reference solution, the
BDD
checker produces a counterexample in the form of a set of function
arguments that cause the test solution to differ from the reference
solution.

The sources are included in ./src/bddcheck. To compile:

```
  unix> cd src/bddcheck
  unix> make clean
  unix> make
```

To use BDDs to check ./src/bits.c for correctness:

```
  unix> cd src
  unix> ./bddcheck/check.pl     # with error messages and
counterexamples
  unix> ./bddcheck/check.pl -g  # compact tabular output with no
error messages
```

Note that check.pl must be run from the parent directory of
./bddcheck.

We've been using this BDD checker instead of btest at CMU for several
years and the code appears to be stable. The main weakness is in the
Perl code that extracts the functions from bits.c. It usually works,
but some things -- such as calls to other functions, or functions
that
don't end with a single brace -- confuse it. So we're reluctant to
make it the default checker for the distributed CS:APP labs. However,
if you have any questions about the correctness of a particular
solution, then this is the authoritative way to decide.

Please send any comments about the BDD checker to
randy.bryant@cs.cmu.edu.