Project 2: Data Operations

- CSci 2021 (010), Spring 2020.
- Released Friday, April 24th.
- Due Monday, May 4th at 11:59 PM

Introduction

We recommend that you do this assignment using the CSE Labs machines. On a CSE Labs machine, you can set up the assignment by extracting the files with the following command:

```
tar xvzf /web/classes/Spring-2020/csci2021/proj/proj2.tar.gz
```

This will create the `proj2-handout` directory, where you will find everything you need.

There is also a copy of the assignment files available from the course web site. However it will not be our priority to provide support for running the assignment on non-CSE-Labs machines. If you do carry out some of your work on other machines, we strongly recommend that you re-test that your solutions work correctly when run on the CSE Labs machines, since your assignment will be graded based on how it runs on the CSE Labs machines.

Testing

The compressed handout contains test programs to help you verify and debug your solutions. For more details, read `README` in the handout.

- `bits.c` - The source file where you write your solutions
- `btest` - Test each puzzle for correct operation
- `dlc` - A compiler that detects whether your `bits.c` program follows the coding rules for each puzzle.
- `fshow` - Displays the (signed and unsigned) floating point value of a hex number equal to the floating point bit representation.
- `ishow` - Displays the (signed and unsigned) integer value of a hex number equal to the integer bit representation.
- `check.pl` - Combination of btest and dlc to "autograde" bits.c.

Grading

Each puzzle will be scored according these parameters:

1. Passing the `btest` test program. Each puzzle has test cases that btest compares with your `bits.c`.
2. Passing the `dlc` compilation test. Each puzzle has specific coding rules, including what operators are legal and how many operators are allowed. Solutions that produce the wrong results or use illegal operators will not receive credit, but solutions that use the legal operators and produce the correct result but use too many operators will get partial credit.
3. The number of points for each puzzle are scaled according to the puzzle rating. Higher-rated puzzles are worth more points than lower-reated puzzles.

70 out of the 100 points for the project will be determined based on automatic testing of your submission, with each point of raw score from the testing scripts being worth two points in the final total. You can earn 1-4 raw points for the correctness of each puzzle solution, based on its difficulty, for a total of 19 raw points or 38 final points. For each correct puzzle solution, you can earn 2 more raw points if your solution uses no more than the limited number of operators we have specified, for a total of 16 raw points or 32 final points.

20 out of the 100 points for the project will be awarded for including the result of check.pl in a comment at the end of your submission. The btest and dlc programs provide important feedback, so we strongly

recommend that you run them early and often during the process of working on your project. If you run btest on its own and one of your solutions is incorrect, it will show you a test case for which your solution gives the wrong answer, which you should find helpful in debugging. And it's also important to use dlc (perhaps via check.pl) to check whether your solutions are following the coding rules. You can't get credit for solutions that don't follow the coding rules, and if your source file is incompatible with dlc we won't be able to grade it at all. To emphasize this importance of this, we are giving you points just for the process of running check.pl and cutting and pasting its report in a comment at the bottom of your source file. You won't get this credit if check.pl doesn't run correctly, but you can get all 20 of these points even if the auto-grading report says you aren't getting any of the correctness or efficiency points (Scaled auto-grading score = 0/70). So you should verify that you're able to run this script from the beginning of when you start working on the project.

The remaining 10 out of the 100 points for the project will be based on the style and readability of your C code. Give variables names that reflect their meaning and usage, and write comments describing the tricks you used to achieve the solutions with unusual or small numbers of operators. But you don't have to write comments describing what the puzzle functions are supposed to do, since we've already written that.

Common Pitfalls

Don't add `#include <stdio.h>` or any external headers to your `bits.c` file, as these header files are not compatible with `dlc`. You can use `printf`, though, because we have provided a declaration for it.

The `dlc` program is based on an older version of the C programming language (C89) than GCC is, so some GCC-supported features will not be accepted by `dlc`. For this assignment your programs will need to be acceptable to both `dlc` and to GCC. The most commonly encountered limitation is that in any given block (set of curly braces), all the declarations must come before any statement that is not a declaration. For instance `dlc` will give an error about the following code:

```
int foo(int x)
{
  int a = x;
  a *= 3;    /* Statement that is not a declaration */
  int b = a; /* ERROR: Declaration not allowed here */
}
```

Instead, you need to either make pre-declare variables, or make every assignment into the declaration of a new variable. For instead either of the following would be OK:

```
int foo(int x)
{
  int a = x;
  int b;
  a *= 3;
  b = a;
}
```

```
int foo(int x)
{
  int a = x;
  int a_times_3 = a * 3;
  int b = a_times_3;
}
```

Submitting

This assignment is due on Monday, May 4th at 11:59 pm. Please review the course syllabus for the late submission policy.

Please submit your solution to this lab on Canvas. You must submit a single .c file that contains all of your code. Please name the file:

`bits_<X500>.c`

Where `<X500>` is the part of your UMN email address before the `@umn.edu`, also the same as your CSE Labs username and the sometimes called an Internet ID or X.500 identifier.

For example, if your email address is `goldy001@umn.edu`, your filename should look like:

`bits_goldy001.c`