CSC3060 Project 1: Data Lab

mingyibao@link.cuhk.edu.cn

Last updated: 2026-01-15 22:19

1 Overview

In this project, you will implement five fundamental arithmetic operations,
`add, subtract, multiply, divide, modulo`,
using only bitwise operations.
Estimated time: 15 hours. Difficulty: Easy to Medium.

2 Project Structure

```
datalab/
├── include/                        (folder for header files)
│   └── data_lab.hpp                (function declarations - provided)
├── src/                            (folder for source files)
│   └── data_lab.cpp                (YOUR IMPLEMENTATION)
└── test/                           (folder for unit tests)
│   └── test_data_lab.cpp           (unit tests - provided)
├── .clang-format                   (configuration files for ClangFormat)
├── .clang-tidy                     (configuration files for Clang-Tidy)
├── .gitignore                      (configuration files for gitignore)
├── CMakeLists.txt                  (build and test configurations - provided)
└── README.md                       (project description)
```

2.1 What We Provide

https://github.com/bmyjacks/CSC3060_data_lab

- `CMakeLists.txt`: Build system (provided as-is). You are recommend to read/modify it.
- `data_lab.hpp`: Function signatures. Do not modify it as we need to link against your library.
- `test_data_lab.cpp`: Unit tests. You are recommended to read it.

2.2 What You Write

You **ONLY** need to modify the following files

- `data_lab.cpp`: Implementations of 5 functions.
- `report.pdf`: Documentation of your approach.

3 Function Requirement

We focus on 32-bit signed integers.

3.1 Signatures

```cpp
int32_t add     (int32_t a, int32_t b)    // return a + b
int32_t subtract(int32_t a, int32_t b)    // return a - b
int32_t multiply(int32_t a, int32_t b)    // return a * b
int32_t divide  (int32_t a, int32_t b)    // return a / b
int32_t modulo  (int32_t a, int32_t b)    // return a % b
```

3.2 Expected Behavior

Align with normal C++ behavior, meaning your result should be the same as if you used those arithmetic operators directly in C++.

Examples:

- Handle negative numbers correctly.
- Overflow behavior matches C++ signed integer overflow.
- For division, truncates toward zero. $(-10 \div 3 = -3, 10 \div 3 = 3)$.
- For modulus, sign follows the dividend. $(-10 \,\%\, 3 = -1, 10 \,\%\, -3 = 1)$.
- Error handling for division/modulus by zero is **NOT REQUIRED**.

4 Restrictions & Rules

You **CAN** use the following operations

- Bitwise operators: `&`, `|`, `^`, `~`, `<<`, `>>`, `&=`, `|=`, `^=`, `~=`, `<<=`, `>>=`
- Control flow (only in `multiply()`, `divide()`, `modulo()`): `if`, `while`, `for`
- Logical operators: `&&`, `||`, `!`
- Function calls to your own functions
- Type casts: `static_cast<uint32_t>()`
- Constants and arithmetic on constants (e.g., `32`, `32 - 1`)
- Local variables
- Write your own helper functions, under the same rules here
- Implement comparison operators using allowed operations

You **CANNOT** use the following operations

- Arithmetic operators on variables: `+`, `-`, `*`, `/`, `%`, `++`, `--`, `+=`, `-=`, `*=`, `/=`, `%=`
- Comparison operators: `<`, `>`, `<=`, `>=`, `==`, `!=`
- Ternary/conditional operator: `?:`
- Control flow in `add()`, `subtract()`
- Arrays, pointers, or macros
- Recursion
- Tricks like using `add(sum, a)` b times in `multiply(a, b)`
- Inline assembly

More on the arithmetic operators: You can use them with constants and control flow variables. However, they cannot be used in other situations. See examples below:

```cpp
constexpr int32_t LEN = 10; // Allowed
std::cout << LEN << LEN + 1; // Allowed

for (int32_t i = 0; i < LEN; i++) { // Allowed
    a = 2 << i; // Allowed
    a = 2 + i; // NOT allowed
    a = add(2, i) // Allowed
}

x = 1, y = 2;
add(x + 1, y); // NOT allowed
add(add(x, 1), y); // Allowed

a = x >> 16; // Allowed
a = x >> 16 - 1; // Allowed
a = x >> LEN - 1; // Allowed
b = -a; // NOT allowed
b = subtract(0, a); // Allowed
```

All other operations/techniques are **NOT ALLOWED**.
If in doubt, ask via email or Piazza!

4.1 Code Quality

Your functions must be human-readable. Consider using formatters like ClangFormat.

4.2 What to write in your report

- Why bitwise operations work for arithmetic.
- How do you come up with such a solution.
- What difficulties do you encounter, and how do you solve them.
- Time complexity for your implementation.
- Thoughts about unit testing.
- Other things you want to talk about.

5 Grading Rubric

Function	Points
`add`	12
`subtract`	12
`multiply`	12
`divide`	12
`modulo`	12
Mystery real-world tasks	30
Report	10
Total	100

- For each function, we provide 12 unit tests. Each counts for 1 point.
- For the Mystery real-world tasks, you don't need to do anything. We will test your function against some famous computation-heavy algorithms like Hashing, Encryption, etc. There are 30 unit tests, each counts for 1 points.
- The Mystery real-world tasks will be released 24 hours after deadline.
- If your implementation passes the provided $5 \times 12 = 60$ unit tests, you should expect at least 60 points for this project.
- We require only the correctness of your result.
- If you use some of the "not allowed" operations in a function, the corresponding point for that function and Mystery real-world tasks will be 0. For example, if you use `+` in `add()`, the point for `add()` and Mystery real-world tasks will be 0. Points for other functions will still be based on number of tests passed.

5.1 Late Policy

- For the first day after the deadline (2026-01-31 00:00 to 2026-01-31 23:59), 10 percent of your actual score will be deducted from your final project 1 score.
- From 2026-02-01 00:00 to 2026-02-01 23:59, you will get 0 points for Mystery real-world tasks.
- After 2026-02-02 00:00, your score will be 0.

Please start this project early. Although it may look easy, you may encounter difficulties that require time for debugging. If you encounter any unexpected issues, email the Teaching Team to request an extension with a valid excuse and proof.

6 Academic Integrity

- **Understand your code**. You must be able to explain every line of code that you write.
- **Proper attribution**. If you use external resources, such as algorithms from textbooks, websites, or friends, or LLMs, cite them in your code with comments, and explicitly mention them in your report. **If you use LLMs, include the conversation in your report.**

7 Interview Policy

- If you claim that you do not use any LLM tools, we will definitely interview you.
- If your report does not match your code, we will interview you.
- We may interview you about other suspicious things.

8 Submission

8.1 Ensure code compiles

```
cmake -B build
cmake --build build
```

If your code need special command to compile, state them in your reports.

8.2 Ensure that all provided tests pass

```
ctest --test-dir build -V
```

If you get `[  PASSED  ] 60 tests.`, you will get 60 points.

8.3 Check files are included

Make sure your folder have these necessary files.

```
YOUR_FOLDER/
├── include/
│   └── data_lab.hpp
├── src/
│   └── data_lab.cpp
├── test/
│   └── test_data_lab.cpp
├── CMakeLists.txt
└── report.pdf
```

8.4 Submit via BlackBoard

Upload all files as a single archive (`.zip, .tar`). Example: `123090003_P1.zip`

Deadline: 2026-01-30 23:59 China Standard Time.

9 Contact & Support

- Responsible TA/USTF: Mingyi BAO
- Email: mingyibao@link.cuhk.edu.cn
- Tutorials: Tuesday 18:00 to 18:50 and 19:00 to 19:50 @ TD208
- Tutorials: Thursday 18:00 to 18:50 and 19:00 to 19:50 @ TA307
- Office Hours: Friday 10:30 - 11:00 @ Old library 103, and other TA/USTFs' OH
- Piazza: https://piazza.com/class/mjygu9ee5nc2vj/#folder=project1