



PROJECT #0 - C++ PRIMER

 **Do not post your project on a public Github repository.**

OVERVIEW

All the programming projects this semester will be written on the [BusTub](#) database management system. This system is written in C++. To ensure that you have the necessary C++ background, we are requiring everyone to complete a simple program assignment to assess your knowledge of basic C++ features. You will not be given a grade for this project, but you are required to complete the project with a perfect score before being allowed to proceed in the course. Any student that is unable to complete this assignment before the deadline will be asked to drop the course.

All of the code in this programming assignment must be written in C++ (specifically C++17). It is generally sufficient to know C++11. If you have not used C++ before, here is a [short tutorial](#) on the language. More detailed documentation of language internals is available on [cppreference](#). [A Tour of C++](#) and [Effective Modern C++](#) are also digitally available from the CMU library.

There are many tutorials available to teach you how to use [gdb](#) effectively. Here are some that we have found useful:

- [Debugging Under Unix: gdb Tutorial](#)
- [GDB Tutorial: Advanced Debugging Tips For C/C++ Programmers](#)
- [Give me 15 minutes & I'll change your view of GDB \[VIDEO\]](#)

This is a single-person project that will be completed individually (i.e. no groups).

Release Date: Aug 25, 2021

Due Date: Sep 12, 2021 @ 11:59pm

PROJECT SPECIFICATION

In this project, you will implement three classes: `Matrix`, `RowMatrix`, and `RowMatrixOperations`. These matrices are simple two-dimensional matrices that must support addition, matrix multiplication, and a simplified [General Matrix Multiply](#) (GEMV) operation.

You will only need to modify a single file: `p0_starter.h`. You can find the file in the [BusTub repository](#) at `src/include/primer/p0_starter.h`.

In this header file, we define the three classes that you must implement. The `Matrix` abstract class defines the common functions for the derived class `RowMatrix`. The `RowMatrixOperations` class uses `RowMatrix` objects to achieve the operations mentioned in the overview above. The function prototypes and member variables are specified in the file. The project requires you to fill in the implementations of all the constructors, destructors, and member functions. Do not add any additional

The instructor and TAs will not provide any assistance on C++ questions. You can freely use Google or StackOverflow to look up about C++ and any errors you encounter.

INSTRUCTIONS

CREATING YOUR OWN PROJECT REPOSITORY

If the below **git** concepts (e.g., repository, merge, pull, fork) do not make sense to you, please spend some time [learning git](#) first.

Follow the [instructions](#) to setup your own PRIVATE repository and your own development branch. If you have previously forked the repository through the GitHub UI (by clicking Fork), PLEASE DO NOT PUSH ANY CODE TO YOUR PUBLIC FORKED REPOSITORY! Make sure your repository is PRIVATE before you **git push** any of your code.

If the instructor makes any changes to the code, you can merge the changes to your code by keeping your private repository connected to the CMU-DB master repository. Execute the following commands to add a remote source:

```
$ git remote add public https://github.com/cmu-db/bustub.git
```

You can then pull down the latest changes as needed during the semester:

```
$ git fetch public
$ git merge public/master
```

SETTING UP YOUR DEVELOPMENT ENVIRONMENT

First install the packages that BusTub requires:

```
$ sudo ./build_support/packages.sh
```

See the [README](#) for additional information on how to setup different OS environments.

To build the system from the commandline, execute the following commands:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

To speed up the build process, you can use multiple threads by passing the **-j** flag to **make**. For example, the following command will build the system using four threads:

```
$ make -j 4
```

TESTING

You can test the individual components of this assignment using our testing framework. We use [GTest](#) for unit test cases. There is one file that contains tests for all three classes:

- **Starter**: `test/primer/starter_test.cpp`

You can compile and run each test individually from the command-line:

```
$ mkdir build
$ cd build
```

You can also run `make check-tests` to run ALL of the test cases. Note that some tests are disabled as you have not implemented future projects. You can disable tests in GTest by adding a `DISABLED_` prefix to the test name.

These tests are only a subset of the all the tests that we will use to evaluate and grade your project. You should write additional test cases on your own to check the complete functionality of your implementation.

 Make sure that you remove the `DISABLED_` prefix from the test names otherwise they will not run!

FORMATTING

Your code must follow the [Google C++ Style Guide](#). We use `Clang` to automatically check the quality of your source code. Your project grade will be zero if your submission fails any of these checks.

Execute the following commands to check your syntax. The `format` target will automatically correct your code. The `check-lint` and `check-clang-tidy` targets will print errors that you must manually fix to conform to our style guide.

```
$ make format
$ make check-lint
$ make check-clang-tidy
```

DEVELOPMENT HINTS

Instead of using `printf` statements for debugging, use the `LOG_*` macros for logging information like this:

```
LOG_INFO("# Pages: %d", num_pages);
LOG_DEBUG("Fetching page %d", page_id);
```

To enable logging in your project, you will need to reconfigure it like this:

```
$ mkdir build
$ cd build
$ cmake -DCMAKE_BUILD_TYPE=DEBUG ..
$ make
```

The different logging levels are defined in `src/include/common/logger.h`. After enabling logging, the logging level defaults to `LOG_LEVEL_INFO`. Any logging method with a level that is equal to or higher than `LOG_LEVEL_INFO` (e.g., `LOG_INFO`, `LOG_WARN`, `LOG_ERROR`) will emit logging information. Note that you will need to add `#include "common/logger.h"` to any file in which you want to make use of the logging infrastructure.

We encourage you to use `gdb` to debug your project if you are having problems.

 Post all of your questions about this project on Piazza. Do not email the TAs directly with questions.

GRADING RUBRIC

In order to pass this project, you must ensure your code follows the following guidelines:

1. Does the submission successfully execute all of the test cases and produce the correct answer?
2. Does the submission execute without any memory leaks?
3. Does the submission follow the code formatting and style policies?

Note that we will use additional test cases to grade your submission that are more complex than the sample test cases that we provide you.

There are no late days for this project.

SUBMISSION

You will submit your implementation to Gradescope:

<https://www.gradescope.com/courses/286490/>

You only need to include the following files with their full path in your submission zip file:

- `src/include/primer/p0_starter.h`

You can verify the contents of your file using the `unzip` command. The output of this command should look something like

```
$ unzip -l project0-submission.zip
Archive:  project0-submission.zip
  Length      Date    Time    Name
-----
   317      2020-08-25  19:25  metadata.yml
    0      2020-08-25  19:25  src/
    0      2020-08-25  19:25  src/include/
    0      2020-08-25  19:25  src/include/primer/
  4465      2020-08-25  19:25  src/include/primer/p0_starter.h
-----
  4782
                   5 files
```

Although you are allowed submit your answers as many times as you like, you should **not** treat Gradescope as your only debugging tool. Most students submit their projects near the deadline, and thus Gradescope will take longer to process th requests. You may not get feedback in a timely manner to help you debug problems. Furthermore, the output from Grades is unlikely to be as informative as the output from a debugger (like `gdb`), provided you invest some time in learning to use

 CMU students should use the Gradescope course code announced on Piazza.


COLLABORATION POLICY

Every student has to work individually on this assignment.

Students are allowed to discuss high-level details about the project with others.

Students are **not** allowed to copy the contents of a white-board after a group meeting with other students.

Students are **not** allowed to copy the solutions from another colleague.

 **WARNING:** All of the code for this project must be your own. You may not copy source code from other students or other sources that you find on the web. Plagiarism **will not** be tolerated. See CMU' [**Policy on Academic Integrity**](#) for additional information.

Last Updated: Jan 07,

