Project 3 – My Matrix

CS 251, Spring 2023

Collaboration Policy

By submitting this assignment, you are acknowledging you have read the collaboration policy in the syllabus for projects. This project should be done individually. You may not receive any assistance from anyone outside of the CS 251 teaching staff.

Late Policy

You are given the grace period between the deadline and the first lecture of the following day (9am). You do not need to let anyone know you are using this grace period. Beyond this period, no late submissions will be accepted.

What to Submit

Submit *main.cpp* and *mymatrix.h* to Gradescope. The *main.cpp* file is for testing your code.
Do not submit any additional files.

Compilation & Execution

We suggest you use the provided `makefile` but if you wish to compile directly, use the 2020 standard for g++.

Grading & Gradescope Environment

We will use Gradescope to grade and compile your program. It is common for C++ programs to "work on my platform" but fail on an autograder such as Gradescope. This is due to logic errors in ***your*** program, not an error with Gradescope. The most common mistake is a memory-related error, e.g. using an uninitialized variable or accessing memory outside the bounds of an array or via an invalid pointer. These errors are hard to find; the tools valgrind and cppcheck can help; note that valgrind will also report memory leaks, ignore those messages (we don't care about memory leaks in this project).

Gradescope is running on Ubuntu Linux, and we are compiling with the 2020 standard using the std flag via g++ with -std=c++20.

Your score on this project is based on two factors:

1. Correctness as determined by your Gradescope submission
2. Manual review by the TAs for your commenting, style, and approach (e.g. quality of testing code).

The entire project is worth 140 points: 100 points for correctness, and 40 points for commenting, style, and approach. Submissions are expected to compile, run, and pass at least some of the test cases and valgrind must detect no errors; do not expect partial credit with regards to correctness if your program fails all the tests, fails to compile, or valgrind has errors.

Example: If you submit to Gradescope and your score is reported as a 0, then that's your correctness score. The only way to raise your correctness score is to re-submit and obtain a higher score by passing one or more test cases. You have unlimited submissions on this project assignment.

Project 3 – My Matrix

CS 251, Spring 2023

$$3 \cdot 0 + 1 \cdot 2 + 0 \cdot 0 = 2$$

$$\begin{bmatrix} 3 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 4 \\ 2 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & \\ & \end{bmatrix}$$

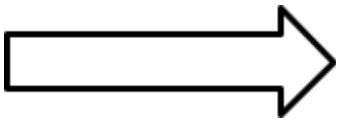
2×3 $\quad$ 3×2 $\quad$ 2×2

Project Summary

In C++, **std::vector<T>** is one of the most commonly used data structures. It's efficient, and grows dynamically as needed. However, it's a one-dimensional data structure. While it's possible to use vector<T> to define two-dimensional structures --- e.g. vector<vector<int>> --- it's awkward because each row is initially empty, requiring you to add columns on a row-by-row basis. There are ways around this, but they are non-obvious or non-intuitive.

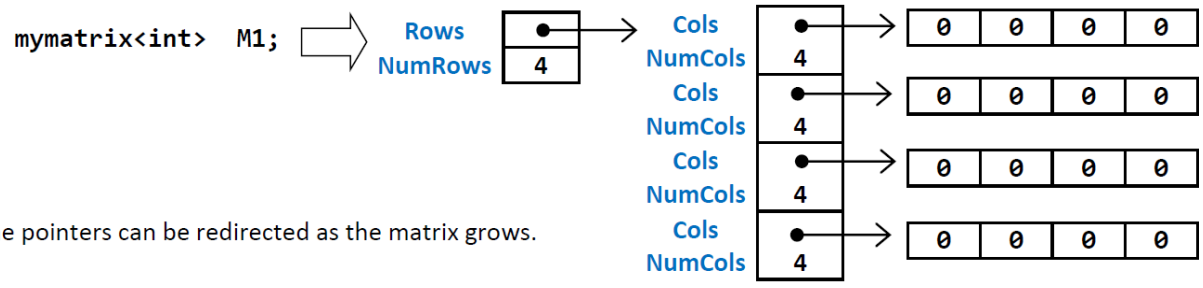
The goal of this project is to define a class **mymatrix<T>** explicitly designed to support a 2D data structure. Like vector<T>, it can grow dynamically in terms of rows and columns. Unlike vector<T>, the use of push_back is not required to add elements. Instead, a matrix is defined to have a given number of rows and columns, and the resulting elements are initialized to C++'s natural default value. For example, the default is a 4x4 matrix:

```
mymatrix<int> M1;
```



0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Keep in mind this is an abstraction, the actual implementation of the matrix is quite different. To allow the matrix to dynamically grow, pointers to C-style arrays are used:



Project 3 – My Matrix

CS 251, Spring 2023

The assignment consists of implementing a set of member functions for class **mymatrix<T>**. The first requirement is that you **must** implement `mymatrix` as discussed on the previous page, and defined in the provided `mymatrix.h` header file. There are lots of possible implementations, but we require that you use this approach. Here are the relevant declarations from "mymatrix.h":

```
template<typename T>
class mymatrix
{
  private:
    struct ROW
    {
      T* Cols; // dynamic array of column elements
      int NumCols; // total # of columns (0...NumCols-1)
    };

    ROW* Rows; // dynamic array of ROWs
    int NumRows; // total # of rows (0...NumRows-1)
```

You are free to add additional member variables to improve the implementation, as well as private helper functions. But you cannot change the overall implementation of a matrix: it must remain a pointer to an array of ROW structures, where each ROW contains a pointer to an array of elements of type T. You cannot switch to a vector-based implementation, nor other data structures.

By default, a matrix is 4x4. Here's the code for the default constructor that creates this 4x4 matrix. You'll want to match this up with the diagram shown previously.

```
  public:
    //
    // default constructor:
    // Called automatically by C++ to construct a 4x4 matrix. All
    // elements are initialized to the default value of T.
    //
    mymatrix()
    {
      Rows = new ROW[4]; // an array with 4 ROW structs:
      NumRows = 4;

      for (int r = 0; r < NumRows; ++r) // initialize each row to have 4 columns:
      {
        Rows[r].Cols = new T[4]; // an array with 4 elements of type T:
        Rows[r].NumCols = 4;

        for (int c = 0; c < Rows[r].NumCols; ++c) // initialize to default value:
          Rows[r].Cols[c] = T{}; // default value for type T:
      }
```

Project 3 – My Matrix

CS 251, Spring 2023

Parts you need to complete are marked with TODO comments. Some are member functions implementing matrix operations, you can use this online calculator or another online matrix calculator to help check your work. You'll note that some member functions are "throwing" exceptions to denote erroneous conditions; this is the standard approach in modern programming languages. We'll discuss this in class at some point, but for now just know that throwing an exception terminates the program, identifying errors quickly.

Testing

An interesting component of this assignment is testing: how do you test your **mymatrix<T>** class? In general, when building software, how do you test it? Normally you run and look at the output, but this is tedious, error-prone, and difficult to repeat. If anything, your eyes get tired. What you want to think about is ways to automate the testing, much like submissions on zyBooks and Gradescope --- push a button and get an answer. We are going to delay the release of our Gradescope submission site so you'll need to do some testing yourself. We are also going to collect and evaluate your testing code as part of your final grade for the project.

In Java, the **JUnit** testing framework is very popular. In the world of C++, there is no single framework that everyone uses, but the C++ Catch framework and the Google Test frameworks are ones we will use in future projects and discuss further in labs and lectures. Both are free with pros and cons. One of the primary benefits of the Catch v1.0 framework is that it is a single header-only (i.e. all you need is the .h file).

If this seems like a lot of work, it is. The general rule of thumb is that it takes as much time to test a piece of software as it does to create it. However, once you write the tests, the advantage is huge: at any moment you can run the tests and get a sense of how well the software is working. [Have you heard of Test-Driven Development (TDD)? This is where you write the tests first, and the software second. This is a popular software development approach.]

Be aware that the results are only as good as the tests you write. If the tests are not very extensive, then the likelihood of the software being correct is low. When you think about testing, you want to think about the following:

1. Have I tested every public function?

2. Since the class is templated, have I tested different types (int, double, string)?

3. Have I tested boundary conditions, e.g. jagged rows? Accessing the first and last row? Accessing the first and last column? Growing a matrix where the new sizes are smaller?

Testing is a topic we'll discuss throughout the semester in both lecture and lab.

Project 3 – My Matrix

CS 251, Spring 2023

Testing Examples

For this project, you should write test functions in `main.cpp`. For example the two below testing functions demonstrate the one way to test the size() function and one way to test an exception throwing function.

```
bool size_test1();
bool exception_test_example();

int main()
{
  int passed = 0;
  int failed = 0;

  if (size_test1())
    passed++;
  else {
    cout << “size_test1 failed”
      << endl;
    failed++;
  }

  cout << “Tests passed: “
    << passed << endl;
  cout << “Tests failed: “
    << failed << endl;
  return 0;
}

bool size_test1()
{
  // creates 4x4 matrix
  mymatrix<int> M;

  if (M.size() == 16)
    return true;
  else
    return false;
}
```

```
bool exception_test_example()
{
  try {
    mymatrix<int>  M1;
    mymatrix<int>  M2;

    M1.growcols(3, 16);

    mymatrix<int> result;

    result = M1 * M2;
    // matrix multiply should throw an exception

    // if we get here
    // no exception was thrown --- error
    cout << "matrix multiply test failed: "
      << "jagged matrix did not throw exception"
      << endl;
    failed++;
  }
  catch (...) {
    passed++;
    // if we get here,
    // exception was thrown --- correct
  }
}
```

Reminder, your test coverage and thoroughness will be graded by the TAs.

Project 3 – My Matrix

CS 251, Spring 2023

Requirements

Note that the TAs will also review for adherence to requirements; breaking a requirement can result in a final score of 0 out of 140. We take all requirements seriously.

1. The implementation of a matrix must follow the diagram shown at the bottom of page 1, and as defined in the provided "mymatrix.h" file. In other words, a matrix must remain a pointer to an array of ROW structures, where each ROW contains a pointer to an array of elements of type T. You cannot switch to a vector-based implementation, nor other data structures.

2. Feel free to define additional variables or functions. However, define them as private member variables and functions. No global or static variables.

3. Your "mymatrix.h" program file must have a header comment with your name and a class overview. Much of this has already been written for you. Likewise, each function must have a header comment above the function, explaining the function's purpose, parameters, and return value (if any). Inline comments should be supplied as appropriate; comments like "declares variable" or "increments counter" are useless. Comments that explain non-obvious assumptions or behavior *are* appropriate.

4. You need to put some effort into testing by writing a "main.cpp" file and submitting that for evaluation along with your "mymatrix.h" file. You can expect your testing code to be evaluated roughly as "bad", "okay", "good", or "excellent". Excellent means you tested every member function in a non-trivial way.

5. **Do not free memory or write a destructor.** Ignore this for now, since freeing memory often triggers hard-to-find pointer errors. We'll worry about freeing memory in future projects; for now, enjoy leaking memory without worry 😀

Citations/Resources

Original assignment design by Joe Hummel, PhD - University of Illinois Chicago.