

Project 3 – Matrix Calculator

Assigned: Feb 7, 2012

Due: Feb 16, 2012, 11:59pm

Introduction. We will extend our previously developed complex calculator to provide for the simple matrix operations add, subtract and multiply (we won't do divide). To implement this, we will create a class called `Matrix` that implements a two-dimensional matrix. We will implement several constructors, including one that populates the matrix with data from a character string. To describe a matrix with a string, we use parenthesis to delineate the rows of the matrix. For example:

`(1, 2, 3), (4, 5, 6), (7, 8, 9)`
would represent the matrix:

```
1 2 3
4 5 6
7 8 9
```

All the rules from previous calculator program apply, such as only a single operator per input line, lines starting with an operator use the prior result at the first operand, etc. We will also use a *Not A Matrix* flag in our `matrix` class to indicate that the matrix is invalid. This would be set when the size of the matrices being added or multiplied are not compatible.

Since we don't know at compile time how many rows or columns a matrix object will have, we must use dynamic memory management to allocate and return memory for the rows and columns. One possible (good) design is to implement a class called `MatrixRow` that is managed by the `Matrix` objects. In the constructor for `Matrix` you should allocate memory for the rows using `new` once you know how many rows there are. You will then need constructors in the `MatrixRow` class to say how many columns there are and allocate memory for the columns, and you will need member functions to read and write the the values in the columns.

Specific Program Requirements.

1. You must define and implement a `Matrix` class, with a constructor with a `string` argument, to construct a matrix with initial contents. In this case the size of the matrix is apparent from the input string. Additionally, you will need a second constructor with two unsigned int's specifying an `m x n` matrix initialized with all zeros (to be used for results in arithmetic expressions).
2. You must define and implement a `MatrixRow` class, with a constructor with at `string` argument to create a row with initial contents. Additionally you will need a second constructor with an `int` argument specifying the number of columns, and initializaing all columns to zero.
3. Since your `Matrix` class allocates memory in the constructor (for the variable number of rows), you *MUST* implement a destructor that frees the memory, plus a copy constructor and an assignment operator. Similarly the `MatrixRow` object allocates memory for the columns, so you will need a copy constructor, assignment operator, and destructor for it as well.
4. You must implement a `IsNaM` function that returns a boolean true/false indicating whether the matrix is *Not a Matrix*.
5. The `Matrix` class and `MatrixRow` class must implement *indexing operator* (`operator[]`) to access individual elements in the matrix. Think carefully about how to do this.
6. All operator overloads must be implemented as **member functions**, and all operator overloads must have `const` for both left-hand-side and right-hand-side.
7. Values within the matrix should be stored as type `Element_t`. This is defined to be an `int` in `matrix.h`, but we could change to a `double` later and not require any code changes.

- Matrix arithmetic should be performed using operator overloading.
- You must implement a `Print()` member function. The printed matrix must have the column values aligned right-justified, such as:

```

25  123   5   0   0
 0    2   3   0   0
 0  999  10   0   0
 0    0   0   1   1
 3    0   0   0   3

```

To make the assignment a bit easier, you can assume that no individual element value has more than 4 digits and all numbers are positive. You might decide to add a `PrintElement(int)` member function in the `MatrixRow` class to handle this.

- You will likely find that your main loop in `matrix-calc.cc` is nearly identical to the main loop in the original `complex-calc.cc` from the earlier program.

Design Philosophy. Your program design should be as simple as possible, but no simpler. ¹

Resources. There are a number of files that are given to you. These are all on the `jinx-login` system, and you will copy these to your own directory (instructions as to how to do this are below).

- A skeleton `matrix-calc-skeleton.cc` program that is a starting point. It contains the subroutine to read a line from standard in and call the string parser. Your main loop should exit when an empty line is encountered (when the string parser returns a count of zero).
- Skeleton `Matrix.h` describes the requirements for the `Matrix` class. You should implement the required functions in `Matrix.cc`.
- `string-parse.h` and `string-parse.cc` are provided for string parsing. Note the *ToElement* conversion function that converts a string to an integer `Element_t`. Also note that **anything inside a pair of parens is not a separator**. Thus, parsing “(1, 2, 3), (4, 5, 6)” with a separator of comma gives two substrings, “(1, 2, 3)” and “(4, 5, 6)”. To reduce those substrings to three individual integers you of course use `RemoveParens` and then `StringParse` again.
- A `Makefile` for building the executable binary.
- A second executable called `test-matrix` runs several canned test cases.
- `test-matrix-out.txt` is the expected results from running `test-matrix`.

Your program can be compiled and tested on any available computing platform that has a C++ compiler. The instructor and TA will compile and test your program on the `jinx` linux systems. Be sure to put your name on the source code in the comments section.

Copying the Project Skeletons

- Log into `jinx-login.cc` using `ssh` and your prism log-in name.
- Copy the files from the `ECE3090` user account using the following command:

```
/usr/bin/rsync -avu /nethome/ECE3090/MatrixCalculator .
```

Be sure to notice the period at the end of the above command.

¹Paraphrased from quote by Albert Einstein

3. Change your working directory to `MatrixCalculator`

```
cd MatrixCalculator
```

4. Copy the provided `matrix-calc-skeleton.cc` to `matrix-calc.cc` as follows:

```
cp matrix-calc-skeleton.cc matrix-calc.cc
```

5. At this point, you can run the `make` program to compile the skeletons. Of course it won't do anything useful since you have not implemented your `Matrix` class.
6. Use `emacs` or `vi` to edit the files as needed to implement your program.
7. You can test your program against a fixed set of inputs in file `input.txt` as follows:

```
./matrix-calc < input.txt
```

8. The expected outputs from this are found in file `output.txt`

Checking for Memory Leaks and Other Memory Problems This type of program with dynamic memory management is particularly difficult to get implemented correctly. Often programs that are *buggy* and mis-manage memory will in fact work properly and produce right answers in some cases. However, we still need to correctly manage all dynamically allocated memory. On the linux systems, there is a nice tool called `valgrind` that helps us identify and fix problems with memory management.

To use `valgrind`, you simply run the `valgrind` program passing as arguments the name of the program to be tested and all parameters to your program. For the matrix calculator, you would likely say:

```
valgrind --tool=memcheck ./matrix-calc < input.txt
```

The output of `valgrind` is sometimes hard to read; so we will go over this in class. We *will* run `valgrind` on your program when grading, and to get a 100% you must get a "Clean Valgrind" (with no errors reported).

Turning in your Project. The system administrator for the `jinx` cluster has created a script that you are to use to turn in your project. The script is called `riley-turnin` and is found in `/usr/local/bin`, which should be in the search path for everyone. From your **home directory** (not the `MatrixCalculator` subdirectory), enter:

```
riley-turnin MatrixCalculator.
```

This automatically copies everything in your `MatrixCalculator` directory to a place that I can access (and grade) it.