Assignment 2: Implementing Matrix ADT

Due: April 08, 2022 by 23:55 a.m.

Assignment 2 is worth 12.5% of your final grade.

1 Introduction

A $m \times n$ matrix is a two-dimensional array with m rows and n columns. Matrices are often used to organize data. We use the notation $M(i, j), 0 \leq i \leq m-1, 0 \leq j \leq n-1$ to refer to the element in row i and column j of matrix M. As an example of data organization, consider the students of UofL and the courses they are currently enrolled in or have completed at UofL. We can represent this information conveniently by listing the students along the rows and courses along the columns as below:

$$M = \left(\begin{array}{c|cccc} & c_0 & c_1 & c_2 & c_3 \\ \hline s_0 & 1 & 0 & 0 & 0 \\ s_1 & 1 & 1 & 0 & 0 \\ s_2 & 0 & 0 & 1 & 0 \\ s_3 & 1 & 1 & 1 & 1 \end{array}\right).$$

where the labels $s_0 \ldots s_3$ represent students and the labels $c_0 \ldots c_3$ represent courses. Then, a value of 0 in M, for example for the entry $M(s_1, c_3)$, can be interpreted as "student s_1 has not completed or currently enrolled in course c_3". Also, a value of 1 in M, for example, for the entry $M(s_1, c_2)$ can be interpreted as "student s_3 has completed or currently enrolled in course c_3. When there is no ambiguity, the labels are not explicitly given and the student-course matrix M can simply be written as

$$M = \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}\right).$$

To refer to an entry of M we just use the row and column numbers. Then $M(0, 3) = 0$ can be interpreted as "student number 0 has not completed or currently enrolled in course number 3".

2 Data Structure for Matrix Abstract Data Type (ADT)

A matrix ADT can be implemented, for example, using a two-dimensional or a one-dimensional array. If the size of the matrix is not known at the creation time or if the size can vary at

runtime, it is reasonable to implement it as a dynamic array. To refer to or access individual matrix elements, one can use a pair of parentheses () which is more common in mathematical writings, or square brackets [ ][ ] common in programming languages.

In this assignment you are asked to use a *one-dimensional dynamic array* to implement the matrix ADT. Since a matrix is a two-dimensional object, we need to find a way to **map** the elements of a two-dimensional array into a one-dimensional array. In other words, we need a formula to identify (i, j)th entry of matrix M, i.e., the element in row i and column j, in its one-dimensional array representation. As it turns out, this mapping can be done in two ways:

1. Row-major mapping.
2. Column-major mapping.

To illustrate, consider the following matrix:

$$M = \begin{pmatrix} 1 & 1 & 2 & 3 \\ 5 & 8 & 13 & 21 \\ 34 & 55 & 89 & 144 \end{pmatrix}.$$

Assume that the rows and columns are labelled from 0. Since M consists of 3 rows and 4 columns, it has room for 12 elements and therefore we need an array of length 12 to represent M. In the row-major mapping we store the elements of row 0 (in increasing column index order) then the elements of row 1 ... and so on. Thus, in the row-major mapping M's elements are stored in a one-dimensional array as shown below,

$$A = \begin{bmatrix} 1 & 1 & 2 & 3 & 5 & 8 & 13 & 21 & 34 & 55 & 89 & 144 \end{bmatrix}.$$

In the column-major mapping the elements of M are placed in a one-dimensional array column-by-column as shown below.

$$A = \begin{bmatrix} 1 & 5 & 34 & 1 & 8 & 55 & 2 & 13 & 89 & 3 & 21 & 144 \end{bmatrix}.$$

In this assignment you will use the row-major mapping. The actual mapping of (i, j)-th element of M, $M(i, j)$, can be expressed by a formula involving the indices i and j. To derive this formula for row-major mapping, recall that the elements of a row are laid out in the order of increasing column indexes and that the elements of row $i - 1$, $i \geq 1$ occur before the elements of row i. Therefore, the (i, j)-th element of M, $M(i, j)$, will occur after the elements of rows 0 to $i - 1$ and the elements in column position 0 to $j - 1$ in row i. This immediately provides the formula we are looking for:

$$M(i, j) = A[i * n + j].$$

In this assignment you are asked to implement the matrix ADT by a `C++` class that has constructors and usual matrix operations such as addition $(+)$, subtraction $(-)$, and multiplication $(*)$. Additionally, your matrix class should allow input and output (standard input/output and file input/output) , a copy constructor, a destructor, an overloaded assignment operator, and overloaded indexing operators . The matrix class definition is provided to you in the next page. Detail specification of the class is given in Section 3.

```
#ifndef MATRIX_H
#define MATRIX_H
#include <iostream>
using namespace std;
class Matrix {
public:
  // Construction
   Matrix(int r = 0, int c = 0, double d = 0.0);

  // Copy construction
   Matrix(const Matrix& m);

   //Type conversion constructon
   Matrix(int r, int n, vector<double>  v);

   // Destructor
   ~Matrix();

   // Index operators
   const double& operator()(int i, int j) const; //to work on const objects
   double& operator()(int i, int j);

   // Copy assignment operator
   Matrix& operator=(const Matrix& m);

   // Compound arithmetic operators
   Matrix& operator+=(const Matrix& x);
   Matrix& operator-=(const Matrix& x);
   Matrix& operator*=(const Matrix& m);
   Matrix& operator*=(double d); // scalar multiplication

   // Output
   void print(ostream& sout) const; //display the matrix onto output stream
                                    // sout neatly
   int ncols() const; //return the number of columns
   int nrows() const; // return the number of rows
private:
    int rows; // number of rows
    int cols;  // number of columns
    double *element; //dynamic array to hold  data
};
// Arithmetic operators are not members
Matrix operator+(const Matrix& l, const Matrix&r); // return   l+r
Matrix operator-(const Matrix& l, const Matrix&r);  // return l-r
Matrix operator*(const Matrix& l, const Matrix&r); //  return l*r
Matrix operator*(double d, const Matrix& r);          // return d*l
Matrix operator*(const Matrix& m, double d);        // return l*d
```

```
// Overloaded stream insertion operator
ostream& operator<<(ostream& out, const Matrix& x);
#endif
```

3 Specification of functions

1. **Constructors.**

   (a) The constructor

   ```
   Matrix::Matrix(int r, int c, double d );
   ```

   accepts three arguments: r being the number of rows, c being the number of columns, and an initialization value d = 0.0 for the contents. If none of the arguments is supplied, the above constructor acts as a default constructor with the value of r and c set to 0.

   (b) The type conversion constructor

   ```
   Matrix::Matrix(int r, int c, vector<double> v );
   ```

   accepts three arguments: r being the number of rows, c being the number of columns, and a vector of doubles v containing r*c double values. It creates a matrix with the specified rows and columns and the values provided in the parameter v.

   (c) **Copy constructor.** The copy constructor creates a new matrix object which is an identical copy of the object passed via parameter.

   ```
   Matrix::Matrix(const Matrix& m)
   ```

   (d) **Destructor.** Releases the dynamic memory held by the matrix object.

   ```
   Matrix::~Matrix() {delete [] element;}
   ```

   (e) **Overloaded assignment operator.** The assignment operator

   ```
   Matrix& Matrix::operator=(const Matrix&);
   ```

   performs "deep copy".

   (f) **The indexed access operator.** The indexed access operator is `()`. For a Matrix object m1, the element in row i and column j, `m1(i,j)`, is returned

   ```
   double& Matrix::operator()(int i, int j)
   ```

   In order to access constant Matrix objects we need another index operator which returns objects by const reference.

   ```
   const double& Matrix::operator()(int i, int j) const
   ```

(g) **Overloaded Arithmetic Operators.** As given in the class definition with their usual meaning.

(h) **Overloaded stream insertion operator**

```
void Matrix::print(ostream& sout) const
```

Member function `print` outputs the matrix object to the output stream `sout` neatly formatted. Minimum requirement is that the matrix is displayed as a rectangular array. The stream insertion operator is overloaded by making a call to `print` member function.

```
//overload << as nonmember
ostream& operator<<(ostream& out, const Matrix& x)
```

In addition to the implementation of the class `Matrix`, write two function templates as specified below.

1. **Write to File.** *Precondition*: vector a has len elements of type T
   Postcondition: The first line of the file contains three integers: nrow and ncol, len, in that order; len elements from vector a are written to the file named data starting from the second line, 10 elements per line.

   ```
   template <class T>
   void writeToFile(vector<T> & a, int len, int nrow, int ncol, const string data)
   ```

2. **Read From File.** *Precondition* The first line contains three integers. The remaining lines contain data; 10 elements per line, except possibly the last line which may contain fewer than 10 elements.
   Postcondition: The three integers in the first line are read into the parameters nrow, ncol, and len, in that order len elements are read from file data, into the vector parameter a

   ```
   template <class T>
   void ReadFile(const vector<T> & a, int &nrow, int &ncol, const string data)
   ```

4 Assignment 4 Tasks

In this assignment you are asked to implement the class `Matrix` as defined, implement the file input/output functions specified in sections 1, 2, and 3, and write a test program to thoroughly test each function you have implemented. The file input/output functions can be included in the test program.

5 What to submit

1. The completed implementation of class `Matrix` in files named `Matrix.h` and `Matrix.cc`.

2. A test program in a file named `test_matrix.cc`, that contains the implementation of the file input/output functions, to thoroughly test each function of the class and file input/output functions.

6 How to submit

There will be a link on CPSC2620 `moodle` page to upload your code. Sample matrix data files will be provided.

7 Grading

The assignment (Assignment 4) will be graded as follows.

1. The program is complete and compiles without errors: 10 points.
2. The class implementation meets the specification: 60 points.
3. The file input/output functions are correct and meets the specification: 10 points.
4. Program is appropriately commented and indented: 10 points.
5. The file `test_matrix.cc` contains code that demonstrates the correct working of each implemented member function: 10 points.

Total: 100 points