Class CMPS 261
Section 001
Problem Programming Assignment #5
Name McKelvy, James Markus
CLID Jmm0468
Due Date 12:30pm December 1, 2005

II. Design Documentation

II.1 System Architecture Description

The object(s) within the program are:
matrix: matrix.h
matrix::iterator: matrix.h

The main objective of the application is to test the implementation of the Matrix Container Class for errors and correctness of the implementation. The application's main driver will not prompt the user, it will only output the various tests to the user for verification that the implementation really "works". After doing so, the program will exit.

II.2 Information about the Objects

II.2.1 Class Information
Name: matrix
Description: Simulates an STL-version of a matrix by using vectors.
Base Class: vector<T>

II.2.2 Class Attributes
Name: iterator
Description: Create a iterator type for the Matrix class based on the M_Iterator class. The user will use this type definition for a Matrix iterator.
Type: typedef M_Iterator
Range of acceptable values: Those conforming to M_Iterator

Name: row_len
Description: The number of rows in the matrix
Type: int
Range of acceptable values: numbers greater than 0

Name: col_len
Description: The number of columns in the matrix
Type: int
Range of acceptable values: numbers greater than 0

Name: myvec
Description: The matrix itself, a Vector of Vectors of T
Type: vector<vector<T> > myvec;

Range of acceptable values: Any of type T

II.2.3 Class Operations

Name: Void Default Constructor
Prototype: matrix<T>(void)
Description: Default constructor to produce an empty Matrix
Precondition(s): None
Postcondition(s): A matrix of vectors is created with default dimensions
Cost Analysis: O(1)
Visibility: public

Name: Default Constructor for m by n matrix
Prototype: matrix<T>(int m, int n)
Description: Default constructor to construct an m by n Matrix
Precondition(s): M and N are supposed to be >= 1
Postcondition(s): A matrix of vectors is created with m by n dimensions
Cost Analysis: O(1)
Visibility: public

Name: Default Constructor for m by n matrix with initial values
Prototype: matrix<T>(int m, int n, T init)
Description: Default constructor to construct m by n matrix where all values are set to the initial value of init
Precondition(s): M and N are supposed to be >= 1, init is a valid value of T
Postcondition(s): A matrix of vectors is created with m by n dimensions, and all initialized to the value of init.
Cost Analysis: O(1)
Visibility: public

Name: Indexing Operator
Prototype: vector<T> &operator[](int loc)
Description: Overloaded [] operator for simulated array access
Returns the int vector< t > row
Precondition(s): The matrix has been initialized with dimensions and/or default values. The value of "loc" is expected to be a valid index in the array (0 <= loc < row_len).
Postcondition(s): A segmentation fault will occur if the value of "loc" is outside of the range. Otherwise, the vector<T> stored at location "loc" in myvec will be returned.
Cost Analysis: O(1)
Visibility: public

Name: At Function
Prototype: T at(int x, int y)
Description: Return the value stored at location x, y
Precondition(s): x and y are assumed to be valid in the matrix
Postcondition(s): returns the value stored at location x, y
Cost Analysis: O(1)

Visibility: public

Name: Back Retrieval Function
Prototype: T back()
Description: Return the last element in the Matrix
Precondition(s): The matrix is assumed to be created.
Postcondition(s): Returns the value located at the back of the matrix
Cost Analysis: O(1)
Visibility: public

Name: Front Retrieval Function
Prototype: T front()
Description: Return the first element in the Matrix
Precondition(s): The matrix is assumed to be created.
Postcondition(s): Returns the value located at the front of the matrix
Cost Analysis: O(1)
Visibility: public

Name: Check If Empty Function
Prototype: bool empty()
Description: Return true if the Matrix is empty, false otherwise
Precondition(s): The matrix is assumed to be initialized
Postcondition(s): Returns true if the matrix is empty, false otherwise
Cost Analysis: O(n)
Visibility: public

Name: Matrix Begin Iterator
Prototype: matrix<T>::iterator begin()
Description: Return iterator to first element
Precondition(s): The matrix is assumed to be initialized
Postcondition(s): Returns an iterator to the beginning of the matrix
Cost Analysis: O(1)
Visibility: public

Name: Matrix End Iterator
Prototype: matrix<T>::iterator end()
Description: Return last iterator
Precondition(s): The matrix is assumed to be initialized
Postcondition(s): Returns an iterator to the end of the matrix
Cost Analysis: O(1)
Visibility: public

Name: Assign Value Function
Prototype:
  void assign(T value, matrix<T>::iterator start, matrix<T>::iterator end)
Description: Assign value to all cells between start and end
Precondition(s): value is a valid T object, start and end are valid
  matrix<T>::iterator's

Postcondition(s): Assigns the value to everything in the matrix between start and end
Cost Analysis: O(n^2)
Visibility: public

Name: Clear All Elements Function
Prototype: void clear()
Description: Erase all elements in the Matrix
Precondition(s): The matrix is assumed to be initialized
Postcondition(s): Erases all elements that are in the matrix
Cost Analysis: O(n)
Visibility: public

Name: Erase One Cell Function
Prototype: void erase(matrix<T>::iterator m)
Description: Erase the contents of the cell indicated by iterator
Precondition(s): The matrix is assumed to be initialized
Postcondition(s): Erases the contents at the location specified by the matrix<T>::iterator m
Cost Analysis: O(1)
Visibility: public

Name: Erase Many Cells Function
Prototype: void erase(matrix<T>::iterator start, matrix<T>::iterator end)
Description: Erase the contents of the cells from start to end
Precondition(s): The matrix is assumed to be initialized
Postcondition(s): Erases the values in the matrix from the iterator start to the iterator end
Cost Analysis: O(n^2)
Visibility: public

II.2.1 Class Information
Name: M_Iterator
Description: Provides an iterator for the Matrix class. The iterator is based on the vector iterator as the Matrix is implemented as a vector of vectors.
Base Class: vector<T>::iterator

II.2.2 Class Attributes
Name: row
Description: The Matrix is implemented using a vector of vectors. We will use two vector iterators to simulate the Matrix iterator. We keep track of the current row with vector< vector< T > > iterator, row, and the current column with vector< T > iterator, col. Together these two uniquely identify a cell in the Matrix.
Type: vector< vector< T > >::iterator
Range of acceptable values: Any type of vector<vector<T> >::iterator

Name: col

Description: The Matrix is implemented using a vector of vectors. We will use two vector iterators to simulate the Matrix iterator. We keep track of the current row with vector< vector< T > > iterator, row, and the current column with vector< T > iterator, col. Together these two uniquely identify a cell in the Matrix.
Type: vector<T>::iterator
Range of acceptable values: Any type of vector<T>::iterator

Name: row_begin
Description: Iterator for the beginning of a row in the matrix
Type: vector<T>::iterator
Range of acceptable values: Any of type vector<T>::iterator

Name: row_end
Description: Iterator for the end of a row in the matrix
Type: vector<T>::iterator
Range of acceptable values: Any of type vector<T>::iterator

Name: col_begin
Description: Iterator for the beginning of a column in the matrix
Type: vector<vector<T> >::iterator
Range of acceptable values: Any of type vector<vector<T> >::iterator

Name: col_end
Description: Iterator for the end of a column in the matrix
Type: vector<vector<T> >::iterator
Range of acceptable values: Any of type vector<vector<T> >::iterator

II.2.3 Class Operations
Name: Dereferencing Operator
Prototype: T operator*();
Description: Simulates the use of an M_Iterator as a pointer by returning the contents of the cell in the Matrix pointed to by the vector<T> iterator.
Precondition(s): The Matrix must have been allocate by one of the constructors.
Postcondition(s): None.
Cost Analysis: O(1)
Visibility: public

Name: Pointer Addition Operator
Prototype: M_Iterator operator+()
Description: An overloading of the + operator to simulate pointer addition. The vector<T> iterator is incremented by int. If the result is larger than the number of columns, then the vector<vector<T> > iterator is incremented and the vector<T> iterator is set to the modulo n element of this row. The new value of col is returned.
Precondition(s): The Matrix must have been allocated by one of the constructors.
Postcondition(s): col will have a new value, row may also have a new value.

Cost Analysis: O(1)
Visibility: public

Name: Auto-increment Pointer Operator
Prototype: M_Iterator operator++()
Description: An overloading of the prefix ++ operator to simulate pointer addition. This function is equivalent to + 1 as expressed in the preceeding function. The new value of col is returned.
Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): col will have a new value, row may also have a new value.
Cost Analysis: O(1)
Visibility: public

Name: Auto-increment Pointer Operator
Prototype: M_Iterator operator++(int)
Description: An overloading of the postfix ++ operator to simulate pointer addition. This function is identical to the previous function and its parameter is totally ignored. The old value of col is returned.
Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): col will have a new value, row may also have a new value.
Cost Analysis: O(1)
Visibility: public

Name: Equality Operator
Prototype: bool operator==(M_Iterator b)
Description: An overloading of the equality operator == to test whether one M_Iterator is equal to another M_Iterator
Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): Returns true if this M_Iterator's values are the same as b's values.
Cost Analysis: O(1)
Visibility: public

Name: Combination Addition and Assignment Operator
Prototype: M_Iterator operator+=(int)
Description: An overloading of the combination += operator to simulate pointer addition. This function is equivalent to + int as express in the Pointer Addition Operator Above. The new value of col is returned.
Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): col will have a new value, row may also have a new value.
Cost Analysis: O(1)
Visibility: public

Name: Pointer Subtraction Operator
Prototype: M_Iterator operator-(int)
Description: An overloading of the - operator to simulate pointer subtraction. The vector<T> iterator is decremented by int. If the result is logically before the first column then the vector<vector<T> > iterator is descremented

and the vector<T> iterator is set to the modulo n element in this row. The new value in col is returned.
Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): col will have a new value, row may also have a new value.
Cost Analysis: O(1)
Visibility: public

Name: Auto-decrement Pointer Operator
Prototype: M_Iterator operator--()
Description: An overloading of the prefix -- operator to simulate pointer subtraction. This function is equivalent to - 1 as expressed in the preceeding function. The old value of col is returned.
Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): col will have a new value, row may also have a new value.
Cost Analysis: O(1)
Visibility: public

Name: Auto-decrement Pointer Operator
Prototype: M_Iterator operator--(int)
Description: An overloading of the postfix -- operator to simulate pointer subtraction. This function is identical to the previous function and its parameter is totally ignored. The new value of col is returned.
Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): col will have a new value, row may also have a new value.
Cost Analysis: O(1)
Visibility: public

Name: Combination Subtraction and Assignment Operator
Prototype: M_Iterator operator-=(int)
Description: An overloading of the combination -= operator to simulate pointer subtraction. This function is equivalent to - int as express in the Pointer Subtraction Operator Above. The new value of col is returned.
Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): col will have a new value, row may also have a new value.
Cost Analysis: O(1)
Visibility: public

Name: The Boolean Comparision Operators
Prototype: Various, see below.
bool operator<(M_Iterator b)
bool operator<=(M_Iterator b)
bool operator>(M_Iterator b)
bool operator>=(M_Iterator b)
bool operator==(M_Iterator b)
bool operator!=(M_Iterator b)
Description: These overloaded comparision operators will perform the appropriate comparision of the operand with the current value of the iterators.

Precondition(s): The Matrix must have been allocated by one of the constructors
Postcondition(s): None.
Cost Analysis: O(1)
Visibility: public

II.3 Information about the Main Application

```
#include<iostream>
#include<vector>
#include<string>
#include "Matrix.h"

using namespace std;

// Name: Print Function
// Prototype: void p(string s)
// Description: Used to print out various strings to the user
// Precondition(s): s is a valid string
// Postcondition(s): Prints s to the screen with a newline
// Cost Analysis: O(1)
void p(string s);

int main(){
   const int m2rows = 4, m2cols = 4;
   const int m3rows = 3, m3cols = 3, m3val = 10;

   // create a void matrix
   p("\ncreating a void matrix: m1");
   matrix<int> m1();

   // create a m2rows by m2cols matrix
   p("\ncreating a matrix with dimensions 4x4: m2");
   matrix<int> m2(m2rows, m2cols);

   // create a m3rows by m3cols matrix with all values initialized to m3val
   p("\ncreating a matrix with dimensions 3x3 with initial values of 10: m3");
   matrix<int> m3(m3rows, m3cols, m3val);

   p("\n *** let's test the [] operator ***");
   p(" * for m2 *");
   for(int i = 0; i < m2rows; i++){
      for(int j = 0; j < m2cols; j++){
         cout << "m2[" << i << "][" << j << "] = " << m2[i][j] << endl;
      }
   }

   p("\n *** let's test the [] operator ***");
   p(" * for m3 *");
   for(int i = 0; i < m3rows; i++){
```

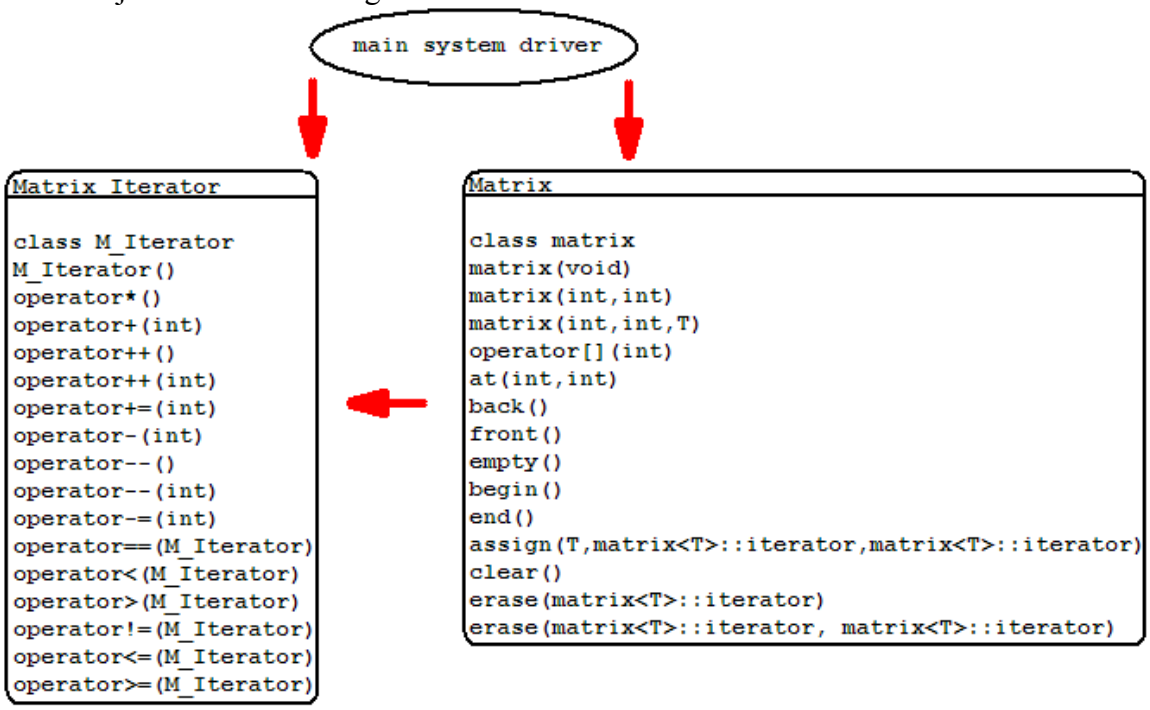
```
    for(int j = 0; j < m3cols; j++){
        cout << "m3[" << i << "][" << j << "] = " << m3[i][j] << endl;
    }
}

p("\n *** let's test the at(x, y) function ***");
p(" * for m2 *");
for(int i = 0; i < m2rows; i++){
    for(int j = 0; j < m2cols; j++){
        cout << "m2.at(" << i << "," << j << ") = " << m2.at(i,j) << endl;
    }
}

p("\n *** let's test the at(x, y) function ***");
p(" * for m3 *");
for(int i = 0; i < m3rows; i++){
    for(int j = 0; j < m3cols; j++){
        cout << "m3.at(" << i << "," << j << ") = " << m3.at(i,j) << endl;
    }
}

p("\n *** let's test the back() and front() functions ***");
cout << "m2.front() = " << m2.front() << " m2.back() = " << m2.back() << endl;
cout << "m3.front() = " << m3.front() << " m3.back() = " << m3.back() << endl;

p("\n *** let's test the empty() function ***");
cout << "m2.empty() = " << m2.empty() << endl;
cout << "m3.emtpy() = " << m3.empty() << endl;

p("\n *** now we will test the begin()/end() iterator functions ***");
cout << "*m2.begin() = " << *m2.begin() << endl;
cout << "*m2.end()   = " << *m2.end() << endl;
cout << "*m3.begin() = " << *m3.begin() << endl;
cout << "*m3.end()   = " << *m3.end() << endl;

p("\n *** now we will test the assign() & begin() functions ***");

p("\nm2.assign(1, m2.begin()+1, m2.begin()+3);");
m2.assign(1, m2.begin()+1, m2.begin()+3);
for(int i = 0; i <= 16; i++)
  cout << "*(m2.begin() + " << i << ") = " << *(m2.begin() + i) << endl;

p("\nm3.assign(15, m3.begin(), m3.begin());");
m3.assign(15, m3.begin(), m3.begin());
for(int i = 0; i <= 9; i++)
  cout << "*(m3.begin() + " << i << ") = " << *(m3.begin() + i) << endl;

p("\n *** now we will test the assign() & end() functions ***");
```

```
    p("\nm2.assign(2, m2.end() - 3, m2.end() - 2);");
    m2.assign(2, m2.end() - 3, m2.end() - 2);
    for(int i = 16; i >= 0; i--)
      cout << "*(m2.end() - " << i << ") = " << *(m2.end() - i) << endl;

    p("\nm3.assign(25, m3.end() - 1, m3.end() - 1);");
    m3.assign(25, m3.end() - 1, m3.end() - 1);
    for(int i = 9; i >= 0; i--)
      cout << "*(m3.end() - " << i << ") = " << *(m3.end() - i) << endl;

    p("\n *** let's erase one cell in m2 ***");
    p("m2.erase(m2.begin() + 1)");
    m2.erase(m2.begin() + 1);
    for(int i = 0; i < m2rows; i++){
        for(int j = 0; j < m2cols; j++){
            cout << "m2[" << i << "][" << j << "] = " << m2[i][j] << endl;
        }
    }

    p("\n *** let's erase a few cells in m3 ***");
    p("m3.erase(m3.begin(), m3.begin() + 1)");
    m3.erase(m3.begin(), m3.begin() + 1);
    for(int i = 0; i < m3rows; i++){
        for(int j = 0; j < m3cols; j++){
            cout << "m3[" << i << "][" << j << "] = " << m3[i][j] << endl;
        }
    }

    p("\nClear all Elements in m2; m3");
    m2.clear();
    m3.clear();

    p("\n *** let's test the empty() function again ***");
    cout << "m2.empty() = " << m2.empty() << endl;
    cout << "m3.emtpy() = " << m3.empty() << endl;

    p("\n\nWe're done.");

    return 0;
}

// Name: Print Function
// Prototype: void p(string s)
// Description: Used to print out various strings to the user
// Precondition(s): s is a valid string
// Postcondition(s): Prints s to the screen with a newline
// Cost Analysis: O(1)
```

```
void p(string s){
  cout << s << endl;
}
```

II.4 Design Diagrams

II.4.1 Object Interaction Diagram



II.4.2 Aggregation Diagram