Solutions for Exam #4 (14/12/2015)

Professors of PRO1

December 15, 2015

General comments

This document only contains possible solutions and no evaluation criteria, since there are no associated "Code Reviews" to this exam.

Since most exercises gave C++ code that ought to be used without any modification, we only reproduce those parts that had to be completed by the students, e.g., the code for the function `obtain_submat` or the code for the function `is_identity`. In some exercises (e.g., *Orthonormal matrices*) the student could use declarations and functions from previous exercises; here, we have indicated where do we use code from other exercises, and written only the "new" code specific to the exercise.

1 Turn #1

1.1 X46890: Obtain submatrix

```
// Pre: 0<=i1<=i2<n and 0<=j1<=j2<m where v is a nxm matrix.
// Post: returns a matrix with dimensions (i2-i1+1)x(j2-j1+1)
// whose elements correspond to the submatrix of v
// with rows between i1 and i2, and columns between j1 and j2.
Mat obtain_submat(const Mat& v,int i1,int j1,int i2,int j2)
{
  Mat R(i2-i1+1, vector<char>(j2-j1+1));
  for (int i = i1; i <= i2; ++i)
      for (int j = j1; j <= j2; ++j)
           R[i - i1][j - j1] = v[i][j];
  return R;
}
```

1.2 X59162: Replace submatrix

```
// Pre: v is a nxm matrix, w is a n'xm' matrix, and
// 0<=i<i+n'<=n and 0<=j<j+m'<=m.
// Post: Returns the result of modifying v by inserting
```

```
// w inside v starting since the position i,j.
Mat replace_submat(Mat v,const Mat& w,int i,int j)
{
  for (int p = i; p < i + int(w.size()); ++p)
       for (int q = j; q < j + int(w[0].size()); ++q)
            v[p][q] = w[p - i][q - j];
  return v;
}
```

1.3 X58992: Central reverse of a matrix

```
// SOLUTION 1
// this solution follows the suggestion given in the statement
// of the problem

// reverses horizontally the nxm matrix by swapping column 0 with
// column m-1, column 1 with column m-2, ... column j with column m-1-j
// if j < m-1-j
void horizontal_reverse(Mat& M) {
       int nrows = M.size(); int ncols = M[0].size();
       for (int j = 0;  < ncols-1-j; ++i)
          for (int i = 0; i < nrows; ++i)
            swap(M[i][j], M[i][ncols-j-1]);

}

// reverses vertically the nxm matrix by swapping row 0 with
// row n-1, row 1 with row n-2, ... row i with row n-1-i
// if i < n-1-i

void vertical_reverse(Mat& M) {
       int nrows = M.size();
       for (int i = 0; i < nrows-1-i; ++i)
            swap(M[i], M[nrows-1-i]); // swap two full rows of M
}

// Returns the result of applying a central reverse to v.
Mat central_reverse(Mat v)
{
  horizontal_reverse(v);
  vertical_reverse(v);
  return v;
}

// SOLUTION 2
// a more compact solution: the result matrix satisfies
```

```
// R[i][j] == v[n-i-1][m-j-1]
// where n = #rows of v and m = #columns of v

// Returns the result of applying a central reverse to v.
Mat central_reverse(Mat v)
{
  int n = v.size(); int m = v[0].size();
  Mat R(n, vector<char>(m));
  for (int i=0; i<n-i-1; i++)
    for (int j=0; j<m-j-1; j++)
        R[i][j] = v[n-i-1][m-j-1];
  return  R;
}
```

1.4 X66668: Reversing submatrices

```
// the declaractions and code of functions and procedures in previous
// exercises (obtain_submat, replace_submat, ...) comes here

int main() {
  Mat v = read_mat();
  int i1, i2, j1, j2;
  bool first = true;
  while (cin >> i1 >> j1 >> i2 >> j2) {
    v = replace_submat(v, central_reverse(obtain_submat(v, i1, j1, i2, j2)),
    write_mat(v);
  }
}
```

2 Turn #2

2.1 X35986: Is identity?

```
// returns true iff the given matrix A is an identity matrix
bool is_identity(const Matrix& A) {
     if (A.size() != A[0].size()) return false; // not a square matrix
     int n = A.size();
     for (int i = 0; i < n; ++i)
         for (int j = 0; j < n; ++j)
             if ((A[i][j] != 1 and i == j) or
                 (A[i][j] != 0 and i != j))
                return false;
     return true;
}
```

2.2 X37157: Transpose

```
// prints the given matrix into the cout; the first line gives
// the number of rows followed by the number of columns, then
// each successive row is printed in a different line, with each
// element of the row separated from the next by a blank space,
// an end-of-line is printed at the end, after the last row of
// the matrix
void print_matrix(const Matrix& M) {
  int m = M.size(); int n = M[0].size();
  cout << m << ' ' << n << endl;
  for (int i = 0; i < m; ++i) {
      cout << M[i][0];
      for (int j = 1; j < n; ++j)
          cout << ' ' << M[i][j];
      cout << endl;
 }
 cout << endl;
}

// returns the transpose of the given matrix
Matrix transpose(const Matrix& M) {
  int nr_rows = M.size(); int nr_cols = M[0].size();
  Matrix Mt(nr_cols, Row(nr_rows));
  for (int i = 0; i < nr_rows; ++i)
      for (int j = 0; j < nr_cols; ++j)
           Mt[j][i] = M[i][j];
  return Mt;
}
```

2.3 X40782: Matrix commands

```
// SOLUTION 1
// in this solution each operations reads its arguments
int main()
{
  char s;
  while (cin >> s) {
    if (s == '+') {
      int n, m;
      cin >> n >> m; Matrix A = read_matrix(n, m);
      cin >> n >> m; Matrix B = read_matrix(n, m);
      // two matrices can be added iff #rows of A == #rows of B
      // and #columns of A == #columns of B
      if (A.size() != B.size() or A[0].size() != B[0].size())
         cout << "The two matrices cannot be added." << endl << endl;
      else
         print_matrix(matrix_sum(A, B));
    } else if (s == '*') {
      int n, m;
      cin >> n >> m; Matrix A = read_matrix(n, m);
      cin >> n >> m; Matrix B = read_matrix(n, m);
      // two matrices can be multiplied iff
      // and #columns of A == #rows of B
      if (A[0].size() != B.size())
         cout << "The two matrices cannot be multiplied." << endl << endl;
      else
         print_matrix(matrix_product(A, B));
   }
 }
}

// SOLUTION 2
// since both operations ('+' and '*') receive two matrices as arguments
// we can read the arguments outside the 'if'; we have also made a few
// other acceptable minor changes, e.g., putting the result of the
// operations in an intermediate matrix C before printing

int main() {
  char s;
  while (cin >> s) {
    int m, n;
    cin >> m >> n; Matrix A = read_matrix(m, n);
    cin >> m >> n; Matrix B = read_matrix(m, n);
    if (s == '+') {
      if (A.size() != B.size())
```

```
            cout<<"The two matrices cannot be added." << endl << endl;
        else if (A[0].size() != B[0].size())
                cout << "The two matrices cannot be added." << endl << endl;
        else {
            Matrix C = matrix_sum(A, B);
            print_matrix(C);
        }
      } else if (s == '*') {
        if (A[0].size() != B.size())
            cout << "The two matrices cannot be multiplied." << endl << endl;
        else {
            Matrix C = matrix_product(A, B);
            print_matrix(C);
        }
      }
    }
}
```

2.4 X43660: Orthonormal matrices

```
// SOLUTION 1
// the declaractions and code of functions and procedures in previous
// exercises (read_matrx, print_matrix, matrix_product, ...) comes here

// returns true iff Q is orthonormal
// Pre: Q is an n x n square matrix, n > 0
bool orthonormal_matrix(const Matrix& Q) {
     return is_identity(matrix_product(Q, transpose(Q)));
}

int main() {
    int m,n;
    bool first = true;
    while (cin >> m >> n) {
        Matrix A = read_matrix(m, n);
        if (first) first = false;
        else cout << ' ';

        if (orthonormal_matrix(A))
         cout << "yes";
        else
         cout << "no";
    }
    cout << endl;
}
```

```
// SOLUTION 2
// an integer square matrix Q is orthonormal iff every row and columns
// contains exactly a 1 or a -1, with all remaining components being 0;
// this allows a more efficient algorithm
// that does not use the subprograms of previous exercises (except read_matr
// the main() function is as above
bool orthonormal_matrix(const Matrix& Q) {
  bool is_orthonormal = true;
  int n = Q.size();
  // check that all rows contain exactly a 1 or a -1, and
  // no entries different from 0, -1, or +1
  for (int i = 0; i < n and is_orthonormal; ++i) {
      int ones_row = 0;
      for (int j = 0; j < n and is_orthonormal; ++j) {
              if (Q[i][j] == 1 or Q[i][j] == -1) ++ones_row;
              if (Q[i][j] > 1 or Q[i][j] < -1 or ones_row > 1)
                 is_orthonormal = false;
      }
      is_orthonormal = is_orthonormal and ones_row == 1;
  }
  // if all rows are correct, check all columns contain
  // exactly a 1 or a -1, no need to check for elements
  // different from 0, -1, or +1, since that was already
  // checked aboved
  if (is_orthonormal) {
   for (int j = 0; j < n and is_orthonormal; ++j) {
      int ones_col = 0;
      for (int i = 0; i < n and is_orthonormal; ++i) {
              if (Q[i][j] == 1 or Q[i][j] == -1) ++ones_col;
              if (ones_col > 1) is_orthonormal = false;
      }
      is_orthonormal = is_orthonormal and ones_col == 1;
   }
  }
  return is_orthonormal;
}
```