Exercise Sheet No.1 Deadline: 29.10.2013, 2 pm

The first exercise session will be on the 5th of November. If you have any questions or need help beforehand, please feel free to send an email.
Please send your solutions to: **ole.klein@iwr.uni-heidelberg.de** before the deadline.

EXERCISE 1 STANDARD LINEAR ALGEBRA CLASSES - OBJECT ORIENTED SOFTWARE DESIGN

The discretisation of a partial differential equation often results in a system of linear algebraic equations. We therefore require the means for basic matrix and vector operations.
In this exercise you will implement the class `Vector` representing a vector and the class `DenseMatrix` representing a standard dense $M \times N$ matrix . Their interfaces should contain at least the following public methods and operators:

For an instance `v` of the class `Vector` the member functions ...

- **`double operator*(const Vector &x) const`** {...}
  should return the **scalar product** of `v` with another vector `x`. We overload this operator instead of defining a standard member function to make the code more readable.
- **`double two_norm() const`** {...}
  should return the **euclidean norm** of `v`.
- **`double two_norm_sqr() const`** {...}
  should return the square of the euclidean norm of `v`.
- **`Vector & scaled_add(const double alpha, const Vector & w)`** {...}
  should add the term `alpha`·`w` to `v` (**scaled vector addition**).
  The *Multiply-Add* operations of modern CPU allow this operation to be implemented with higher efficiency compared to the separated computation of scalar multiplication and vector sum. The function should return a self reference.
- **`Vector & operator*(const double alpha)`** {...}
  should multiply `v` by a scalar `alpha` and return a self reference.

For an instance `A` of the class `DenseMatrix` the member functions ...

- **`double & operator()(int row, int column)`** {...}
  should return a reference to the matrix element at the position defined by `row` and `column`, i.e. `A(row,column)`. Again, we overload the `()` operator instead of defining a standard member function to improve the readability of the user's code.
- **`Vector operator*(const Vector &x) const`** {...}
  should return a `Vector`-type variable that is the product of the matrix `A` with the vector `x`.
- **Direct Gauss solver:**
  `void gauss_solver(Vector & x, const Vector & b) const` {...}
  should solve the linear equation system defined by the matrix `A` and the right-hand side vector `b` using the standard Gauss elimination algorithm. The solution is to be stored in the vector `x`. Of course, this function can be implemented for square matrices only.

Furthermore, both classes **have to define appropriate constructors, destructors, copy constructors and assignment operators**. It is also mandatory to overload the standard stream operator for both classes to allow convenient standard output. Hence define (global) functions:

```
std::ostream & operator << ( std::ostream & os, const Vector & x )
std::ostream & operator << ( std::ostream & os, const DenseMatrix & x )
```

Hints: You may optionally derive `Vector` from the STL container type `std::vector<double>` to simplify the implementation (then the standard copy constructor, destructor and assignment operator will work fine and no additional implementation is necessary). If you represent the matrix data by a `std::vector<double>` object of size $M \times N$, then the same holds for the matrix class.

A raw example framework is given in the *incomplete* header files `Vector.hh` and `DenseMatrix.hh` which you can find on the lecture's website.

`Vector.hh`

```
class Vector : public std::vector<double>
{

  public:
  Vector(const int size) : std::vector<double>(size)
  {}

  // Interfaces ...
};
```

`DenseMatrix.hh`

```
class DenseMatrix
{
  private:
  std::vector<double> data;
  int rows, cols;

  public:
  DenseMatrix(const int rows_, const int cols_, const double def_val) :
  data(rows_*cols_, def_val),
  rows(rows_), cols(cols_)
  {}

  double & operator()(const int row, const int col)
  {
    return data[row * cols + col];
  }

  // Interfaces ...
};
```

You are free to extend the interface of both classes with additional functionality (e.g. infinity norm, iterators, ...). It is allowed to keep all the classes' implementations inside the header files. To see if your implemented classes work properly, a version of a main program `testprogramm.cc` is provided that tests the two classes and their member functions. You can add appropriate matrices and vectors. Please add enough comments to document your code.

5 Points

APPENDIX (REVISION OF GAUSS ELIMINATION):

Gauss Elimination (without pivoting):

for ($k = 1; k < n; k = k + 1$) **do**
  Find $r \in$ {k,...,n} such that $a_{rk} \neq 0$
  and swap rows k and r {require that $a_{kk} \neq 0$ holds}
  for ($i = k + 1; i \leq n; i = i + 1$) **do**
    $q_{ik} = a_{ik}/a_{kk}$;
    for ($j = k + 1; j \leq n; j = j + 1$) **do**
      $a_{ij} = a_{ij} - q_{ik} \cdot a_{kj}$;
    end for
    $b_i = b_i - q_{ik} b_k$;
  end for

end for

This algorithm overwrites matrix entries and right hand side values and it is not numerically stable with regard to floating point errors. The second problem is solved by implementing pivoting.

Gauss Elimination (with row pivoting):

for ($k = 1$; $k < n$; $k = k + 1$) **do**
  Find $r \in$ {k,...,n} such that $|a_{rk}|$ is maximal
  and swap rows k and r
  if ($a_{kk} = 0$) **then**
    STOP, Matrix is singular;
  end if
  for ($i = k + 1$; $i \leq n$; $i = i + 1$) **do**
    $q_{ik} = a_{ik}/a_{kk}$;
    for ($j = k + 1$; $j \leq n$; $j = j + 1$) **do**
      $a_{ij} = a_{ij} - q_{ik} \cdot a_{kj}$;
    end for
    $b_i = b_i - q_{ik} b_k$;
  end for
end for

Row pivoting is most effective when the matrix rows are scaled such that the absolute sum of all rows (p-norm for $p = 1$) are equal. **Hint:** Do not swap the rows in memory but use an array of indices defining the order of the rows and swap indices!
Further information may be found in *P. Bastian (2009) - Numerische und Stochastische Grundlagen der Informatik* (`http://conan.iwr.uni-heidelberg.de/teaching/scripts/numstoch-article.pdf`).

Useful links and books:

- C++ Course: `http://www.vias.org/cppcourse`
- C/C++ Reference: `http://www.cppreference.com`
- Bjarne Stroustrup: The C++ Programming Language, 3rd ed., Addison-Wesley Longman, 1997