```cpp
/*****************************************************************************
File name: Matrix.cpp
Description: Class Matrix
*****************************************************************************/

#include "matrix.h"
#include <vector>
#include <cmath>
#include <fstream>
#include <sstream>

using namespace std;

Matrix::Matrix(double** items, int m, int n)
{
    rowNum = m;
    colNum = n;
    item = new double[m*n];
    for (int i = 0; i < rowNum; i++)
    {
        for (int j = 0; j < colNum; j++)
            item[i*colNum+j] = items[i][j];
    }
}


Matrix::Matrix(int m, int n)
{
    if (m < 0 || n < 0)
    {
        cout << "Error range of m or n!\n";
        return;
    }
    rowNum = m;
    colNum = n;
    item = new double[m*n];
    for (int i = 0; i < m*n; i++)
    {
        item[i] = 0;
    }
}

Matrix::Matrix(double* items, int m, int n)
{
    rowNum = m;
    colNum = n;
    item = new double[m*n];
    for (int i = 0; i < colNum*rowNum; i++)
    {
        item[i] = items[i];
    }
}

Matrix::Matrix(int n)
{
    rowNum = colNum = n;
    item = new double[n*n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
                set(i, j, 1.0);
            else
                set(i, j, 0);
        }
    }
}

Matrix::Matrix(const Matrix &M)
{
    colNum = M.colNum;
    rowNum = M.rowNum;
```

```
74      // should not use the same pointer here
75      item = new double[colNum*rowNum];
76      for (int i = 0; i < colNum*rowNum; i++)
77      {
78          item[i] = M.item[i];
79      }
80  }
81
82  Matrix& Matrix::operator=(const Matrix & M)
83  {
84      colNum = M.colNum;
85      rowNum = M.rowNum;
86  //    if (item != nullptr) delete[] item;
87      item = new double[colNum*rowNum];
88      for (int i = 0; i < colNum*rowNum; i++)
89      {
90          item[i] = M.item[i];
91      }
92      return *this;
93  }
94
95  Matrix::~Matrix()
96  {
97      delete[] item;
98  }
99
100 double Matrix::get(int i, int j) const
101 {
102     return item[i*colNum + j];
103 }
104
105 void Matrix::set(int i, int j, double value)
106 {
107     item[i*colNum + j] = value;
108 }
109
110 void Matrix::RowSwap(int i, int j, double multiply)
111 {
112     if (j == -1)
113     {
114         for (int k = 0; k < colNum; k++)
115         {
116             set(i, k, multiply*get(i, k));
117         }
118     }
119     else
120     {
121         for (int k = 0; k < colNum; k++)
122         {
123             set(j, k, multiply*get(i, k) + get(j, k));
124         }
125     }
126 }
127
128 void Matrix::RowSwap(int i, int j)
129 {
130     Matrix _copy = *this;
131     for (int k = 0; k < colNum; k++)
132     {
133         double swap = _copy.get(j, k);
134         set(j, k, _copy.get(i, k));
135         set(i, k, swap);
136     }
137 }
138
139 Matrix Matrix::Trans() const
140 {
141     Matrix _copy = *this;
142     _copy.rowNum = this->colNum;
143     _copy.colNum = this->rowNum;
144     for (int i = 0; i < _copy.rowNum; i++)
145     {
146         for (int j = 0; j < _copy.colNum; j++)
```

```
147             {
148                 _copy.set(i, j, get(j, i));
149             }
150         }
151         return _copy;
152     }
153 
154     int Matrix::getRowNum() const
155     {
156         return rowNum;
157     }
158 
159     int Matrix::getColNum() const
160     {
161         return colNum;
162     }
163 
164     ostream& operator <<(ostream &os, const Matrix &m)
165     {
166         for (int i = 0; i < m.rowNum; i++)
167         {
168             for (int j = 0; j < m.colNum; j++)
169                 os << std::setw(10) << std::fixed << std::setprecision(12) << m.get(i,
                    j) << " ";
170             os << "\n";
171         }
172         os.flush();
173         return os;
174     }
175 
176     Matrix Matrix::operator +(const Matrix &m)
177     {
178         if (m.colNum != colNum || m.rowNum != rowNum)
179             return *this;
180         Matrix _copy = *this;
181         for (int i = 0; i < rowNum; i++)
182         {
183             for (int j = 0; j < colNum; j++)
184             {
185                 _copy.set(i, j, get(i, j) + m.get(i, j));
186             }
187         }
188         return _copy;
189     }
190 
191     Matrix Matrix::operator -(const Matrix &m)
192     {
193         if (m.colNum != colNum || m.rowNum != rowNum)
194             return *this;
195         Matrix _copy = *this;
196         for (int i = 0; i < rowNum; i++)
197         {
198             for (int j = 0; j < colNum; j++)
199             {
200                 _copy.set(i, j, get(i, j) - m.get(i, j));
201             }
202         }
203         return _copy;
204     }
205 
206     Matrix Matrix::operator *(const double f)
207     {
208         Matrix _copy = *this;
209         for (int i = 0; i < rowNum; i++)
210         {
211             for (int j = 0; j < colNum; j++)
212             {
213                 _copy.set(i, j, get(i, j)*f);
214             }
215         }
216         return _copy;
217     }
218 
```

```cpp
Matrix Matrix::operator *(const Matrix &m)
{
    if (colNum != m.rowNum)
    {
        cout << "can't multiply!";
        return *this;
    }
    Matrix _copy(rowNum, m.getColNum());
    for (int i = 0; i < rowNum; i++)
    {
        for (int j = 0; j < m.colNum; j++)
        {
            double sum = 0;
            for (int k = 0; k < m.rowNum; k++)
            {
                sum += get(i, k)*m.get(k, j);
            }
            _copy.set(i, j, sum);
        }
    }
    return _copy;
}

Matrix Matrix::Inverse()
{
    Matrix _copy = *this;
    // change result
    Matrix result(colNum);
    if (colNum != rowNum)
    {
        cout << "can't inverse!" << endl;
        return *this;
    }
    for (int i = 0; i < rowNum; i++)
    {
        int MaxRow = i;
        // find max absolute number in row i, change it with col i
        double max = abs(_copy.get(i, i));
        for (int j = i; j < colNum; j++)
        {
            if (abs(_copy.get(j, i))>max)
            {
                max = abs(_copy.get(j, i));
                MaxRow = j;
            }
        }
        // change row j with row i
        if (MaxRow != i)
        {
            result.RowSwap(i, MaxRow);
            _copy.RowSwap(i, MaxRow);
        }

        double r = 1.0 / _copy.get(i, i);
        _copy.RowSwap(i, -1, r);
        result.RowSwap(i, -1, r);

        for (int j = 0; j < rowNum; j++)
        {
            if (j == i)
                continue;
            r = -_copy.get(j, i);
            _copy.RowSwap(i, j, r);
            result.RowSwap(i, j, r);
        }
    }
//    result.FlowOver();
    return result;
}

void Matrix::FlowOver()
{
    for (int i = 0; i < rowNum; i++)
```

```
292         {
293             for (int j = 0; j < colNum; j++)
294             {
295                 if (abs(get(i, j)) <= OVERFLOWED)
296                     set(i, j, 0);
297             }
298         }
299     }
300 
301     Matrix Matrix :: getSubMatrix(int startRow, int endRow, int startColumn, int
        endColumn)
302     {
303         Matrix _copy = *this;
304         Matrix subMatrix( endRow - startRow + 1, endColumn - startColumn + 1 );
305 
306         for ( int row = startRow; row <= endRow; ++row )
307         {
308             for ( int column = startColumn; column <= endColumn; ++column )
309                 subMatrix.set(row - startRow, column - startColumn,_copy.get(row,
                    column) );
310         }
311 
312         return subMatrix;
313 
314     }
315 
```