```cpp
/*************************************************************************
File name: Main.cpp
Description: Main Function
*************************************************************************/

#include <iostream>
#include <sstream>
#include <stdio.h>
#include "csv.h"
#include "matrix.h"
#include "CGM.h"
#include "string_to_double.h"
#include "readData.h"
#include "cal_mean.h"
#include "cal_cov.h"

using namespace std;

int  main (int argc, char *argv[])
{
    int numberAssets = 83;                      // set number of assets as 83
    int numberReturns = 700;                    // set number of all days as 700
    int insReturns = 100;                       // set in-sample window size as 100
    int oosReturns = 12;                        // set out_of_sample window size as 12

    // Dynamic Array
    double **retMatrix = new double*[numberAssets];     // matrix to store return
    double *meanMatrix = new double[numberAssets];      // matrix to store mean
    double **covMatrix = new double*[numberAssets];     // matrix to store covariance

    // allocate memory
    for(int i = 0; i < numberAssets; i++)
    {
        retMatrix[i] = new double[numberReturns];
        covMatrix[i] = new double[numberAssets];
    }

    // read the data
    string fileName = "asset_returns.csv";

    // retMatrix[i][j] stores the value of asset i, return j
    readData(retMatrix, fileName);

    // set precision
    cout << fixed << setprecision(4);

    // ---------------------------------------------------------------------------
    // Parameter Estimation

    for(int cnt = 0; cnt < 20; cnt++ )
    {
        double targetReturn = cnt / 200.0; // set target return range from 0 to 0.1 (split into 20 parts)
        cout << "\nNo." << cnt + 1 << " target return : " << targetReturn << endl;

        // ready for output
        stringstream outFileName;
        outFileName << "out_" << targetReturn << ".csv";
        FILE *outFile = fopen(outFileName.str().c_str(), "w");
        int numberWins = 0; // flag to check if oos beat ins

        for(int startDay = 0; startDay < numberReturns-insReturns; startDay += oosReturns)
        {

            // calculate the average return
            cal_mean(meanMatrix, retMatrix, numberAssets, insReturns, startDay);

            // calculate covariance matrix
            cal_cov(covMatrix, meanMatrix, retMatrix, numberAssets, insReturns, startDay);

            // initialize input for optimization
            Matrix Q(numberAssets+2,numberAssets+2);
            for (int i = 0; i < numberAssets; i++)
                for (int j = 0; j < numberAssets; j++)
                    Q.set(i,j,covMatrix[i][j]);

            for (int i = 0; i < numberAssets; i++)
            {
                Q.set(numberAssets,i,-meanMatrix[i]);
                Q.set(numberAssets+1,i,-1);
                Q.set(i,numberAssets,-meanMatrix[i]);
                Q.set(i,numberAssets+1,-1);
            }

            Matrix x0(numberAssets+2,1);
            for (int i = 0; i < numberAssets; i++)
                x0.set(i,0,1./numberAssets);            // initial weights

            Matrix b(numberAssets+2,1);
            b.set(numberAssets,0,-targetReturn);        // set target return of portfolio (rp)
            b.set(numberAssets+1,0,-1);

            // Conjugate Gradient Method to get optimized weights
            x0 = CGM(x0,Q,b);

            // ---------------------------------------------------------------------
            // Backtesting
            double mean_oos = 0;
            double cov_oos = 0;

            cout << "startDay : " << startDay << "\t\t";
            cal_mean(meanMatrix, retMatrix, numberAssets, oosReturns, startDay + insReturns);
            cal_cov(covMatrix, meanMatrix, retMatrix, numberAssets, oosReturns, startDay + insReturns);

            for(int i = 0; i < numberAssets; i++)
                mean_oos += x0.get(i,0) * meanMatrix[i];
            cout << "mean_oos = " << mean_oos << "\t";

            // turn Array into Matrix class
            Matrix covMatrix_oos(covMatrix,numberAssets,numberAssets);
            Matrix w = x0.getSubMatrix(0,numberAssets-1,0,0);
            cov_oos = (w.Trans() * covMatrix_oos * w).get(0,0);
            cout <<"cov_oos = " << cov_oos << endl;

            // ---------------------------------------------------------------------
            // Performance Evaluation
            if (mean_oos > targetReturn) numberWins++;

            // ---------------------------------------------------------------------
            // output results
            fprintf(outFile,"%f,%f\n", mean_oos, cov_oos);
        }
        cout << "win Ratio : " << numberWins*1.0 / ( (numberReturns-insReturns)/oosReturns ) << endl;
        fclose(outFile);
    }

    // free memory
    for(int i = 0; i < numberAssets; i++)
    {
        delete[] retMatrix[i];
        delete[] covMatrix[i];
    }
    delete[] retMatrix;
    delete[] covMatrix;
    delete[] meanMatrix;

    return 0;
}
```