```cpp
//
// 7_2.cpp
//

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <windows.h>

const int N = 100;  // 行列の行数
const double EPS = 1e-6;    // ピボットが 0 かどうか判定する値
typedef double Matrix[N][N+1];  // 行列の型を定義

// csv ファイルから A_mat と b_vec を読み込み、拡大行列を作る関数
void read_A_b(char* A_file, char* b_file, Matrix A1, Matrix A2){
    FILE* A_fp = fopen(A_file, "rt");   // A_mat の入力ファイルポインタ
    FILE* b_fp = fopen(b_file, "rt");   // b_vec の入力ファイルポインタ

    if(!A_fp){  // A_mat が開けなかったらエラー
        fprintf(stderr, "Can't open file %s\n", A_file);
        exit(2);
    }

    if(!b_fp){  // b_vec が開けなかったらエラー
        fprintf(stderr, "Can't open file %s\n", b_file);
        exit(3);
    }

    for(int i=0; i<N; i++){
        for(int j=0; j<N; j++){
            fscanf(A_fp, "%lf,", &A1[i][j]);    // A_mat は 1~N 列目
            A2[i][j] = A1[i][j];
        }
        fscanf(b_fp, "%lf,", &A1[i][N]);    // b_vec は N+1 列目
        A2[i][N] = A1[i][N];
    }
    fclose(A_fp);   // 入力ファイル(A_mat)を閉じる
    fclose(b_fp);   // 入力ファイル(b_vec)を閉じる
}

// 最大値を検索し、その行数を返す関数
int search_max(Matrix A, int k){
    double max = 0.0;   // 最大値を格納する変数
    int max_row = k;    // 最大値がある行数を格納する変数
    for(int i=k; i<N; i++){
        double val = fabs(A[i][k]);
        if(val>EPS && val>fabs(max)){
            max = A[i][k];
            max_row = i;
        }
    }
    return max_row; // 最大値がある行数を返す
}

// 行を入れ替える関数
void change_row(Matrix A, int i, int max_row){
    for(int j=0; j<N+1; j++){
        double tmp = A[i][j];   // 一時的に格納する変数
        A[i][j] = A[max_row][j];
        A[max_row][j] = tmp;
    }
}

// ガウス・ジョルダン法
int gauss_jordan(Matrix A){
    int rankcount = 0;
    for(int i=0; i<N; i++){
        int max_row = search_max(A, i); // 最大値がある行数を検索
        if(fabs(A[i][i]) < EPS){
            for(int j=i+1; j<N; j++){
                if(fabs(A[j][i]) > EPS){
                    change_row(A, i, j);
                break;
                }
            }
        }
```

```
        double pivot = A[i][i]; // ピボットを定義
        if(fabs(pivot) > EPS){ // ピボットが 0 でなければ
            for(int j=i; j<N+1; j++){   // 各列をピボットで除算
                A[i][j] /= pivot;
            }
            for(int k=0; k<N; k++){ // 掃き出し計算
                if(k == i){
                    continue;
                }
                double tmp = A[k][i];   // 一時的に値を保存する変数
                for(int j=i; j<N+1; j++){
                    A[k][j] -= tmp * A[i][j];
                }
            }
        }else{
            rankcount++;    // ピボットが 0 であれば1つランク落ち
        }
    }
    return N - rankcount;
}

// 前進消去
int forward_elimination(Matrix A){
    int rankcount = 0;
    for(int i=0; i<N; i++){
        int max_row = search_max(A, i); // 最大値がある行数を検索
        if(i != max_row){
            change_row(A, i, max_row);  // 最大値がある行数と行を入れ替え
        }
        double pivot = A[i][i];
        if(fabs(pivot) > EPS){ // ピボットが 0 でなければ
            for(int j=i; j<N+1; j++){ // 各列をピボットで除算
                A[i][j] /= pivot;
            }
            for(int k=i+1; k<N; k++){   // 掃き出し計算
                double tmp = A[k][i];   // 一時的に値を保存する変数
                for(int j=i; j<N+1; j++){
                    A[k][j] -= tmp * A[i][j];
                }
            }
        }else{
            rankcount++;    // ピボットが 0 であれば1つランク落ち
        }
    }
    return N - rankcount;
}

// 後退代入
void backward_substitution(Matrix A, int rank){
    if(rank < N){   // 行数と階数が一致していれば後退代入
        return;
    }
    for(int i=N-2; i>=0; i--){
        for(int j=N-1; j>i; j--){
            A[i][N] -= A[i][j]*A[j][N];
        }
    }
}

// ガウス消去法
void gauss_elimination(Matrix A, int& rank){
    rank = forward_elimination(A);  // 前進消去
    backward_substitution(A, rank); // 後退代入
}

// 行列のコピー
void copy_matrix(Matrix A, Matrix copyA){
    for(int i=0; i<N; i++){
        for(int j=0; j<N+1; j++){
            copyA[i][j] = A[i][j];
        }
    }
}

// ガウスジョルダン法およびガウス消去法の計算時間を求める関数
void check_culc_time(int culc_num, Matrix A1, Matrix A2){
    DWORD time1_start, time1_end, time2_start, time2_end;   // 計算時間を格納する変数を定義
```

```
    int rank1 = 0;  // 階数を定義
    int rank2 = 0;  // 階数を定義

    Matrix copy_A1 = {0.0};     // 拡大行列を定義
    Matrix copy_A2 = {0.0};     // 拡大行列を定義

    // ガウスジョルダン法の計算時間
    time1_start = timeGetTime();
    for(int i=0; i<culc_num; i++){
        copy_matrix(A1, copy_A1);
        rank1 =  gauss_jordan(copy_A1);      // ガウスジョルダン法で連立方程式の解を計算
    }
    time1_end = timeGetTime();

    // ガウス消去法の計算時間
    time2_start = timeGetTime();
    for(int i=0; i<culc_num; i++){
        copy_matrix(A2, copy_A2);
        gauss_elimination(copy_A2, rank2);  // ガウス消去法で連立方程式の解を計算
    }
    time2_end = timeGetTime();

    printf("Number of Repetitions : %d\n",culc_num);
    printf("\n----- Gauss Jordan method -----\n");
    for(int i=0; i<N; i++){
        printf("x[%d] = %6.3f\n", i+1, copy_A1[i][N]);
    }
    printf("Calculation time : %ld [ms]\n", time1_end-time1_start);
    printf("\n----- Gauss Elimination method -----\n");
    for(int i=0; i<N; i++){
        printf("x[%d] = %6.3f\n", i+1, copy_A2[i][N]);
    }
    printf("Calculation time : %ld [ms]\n", time2_end-time2_start);
}

// メイン関数
int main(int argc, char* argv[]){
    if(argc!=3){    //実数時引数が２個なかったらエラーとする
        fprintf(stderr, "Usage: %s inputfile outputfile\n", argv[0]);
        exit(1);
    }
    char* A_file = argv[1]; //パラメータの１番目を入力ファイル名
    char* b_file = argv[2]; //パラメータの２番目を入力ファイル名
    Matrix A1 = {0.0};      // 拡大行列を定義
    Matrix A2 = {0.0};      // 拡大行列を定義
    read_A_b(A_file, b_file, A1, A2);   // csv ファイルから A_mat と b_vec を読み込み、拡大行列を作成

    int culc_num = 1000;    // 反復回数
    check_culc_time(culc_num, A1, A2);

    return 0;
}
```