```c
// 6_1.cpp
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const int N = 4;             // 連立方程式の変数数
const double EPS = 1e-6;     // ゼロとみなす閾値
const int STR_LEN = 200;     // 読み込みバッファの長さ

typedef double Matrix[N][N + 1];  // 行列の型を定義

// 行列の表示
void print_matrix(const Matrix A) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j <= N; j++) {
            if (j == N) {
                printf(" |%8.3f", A[i][j]);
            } else {
                printf("%8.3f", A[i][j]);
            }
        }
        printf("\n");
    }
    printf("\n");
}

// 行列をファイルから読み込む
int scan_mat(FILE* ifp, Matrix A) {
    char line[STR_LEN]; // 読み込まれた文字列の保存先

    for (int i = 0; i < N; i++) {
        if (fgets(line, STR_LEN, ifp) == NULL) {
            fprintf(stderr, "Error: Not enough rows in input file.\n");
            return 0;
        }

        char* ptr = line;   // strtod()で使うために、文字列の先頭アドレスをポインタptrに保存。
                            // C言語では、配列名（例えば line）は自動的に先頭要素のポインタに変換されるという仕様がある。&line[0]と同じ。
        for (int j = 0; j <= N; j++) {
            A[i][j] = strtod(ptr, &ptr);    // strtod(ptr, &ptr)は現在のptrの位置からdouble型の数値を1つ読み取る。
                                            // 同時にptrを次の数値の位置に自動で更新する。
        }
    }
    return 1;
}

// ピボット選択：列の絶対値が最大の行と入れ替える
int partial_pivot(Matrix A, int pivot_row) {
    int max_row = pivot_row;
    double max_val = fabs(A[pivot_row][pivot_row]);

    for (int i = pivot_row + 1; i < N; i++) {
        if (fabs(A[i][pivot_row]) > max_val) { // double型の絶対値の取得にはfabs()を使用する。abs()でないことに注意。
            max_val = fabs(A[i][pivot_row]);
            max_row = i;
        }
    }

    if (max_val < EPS) {
        printf("Error: Singular matrix (no suitable pivot).\n");
        return 0;
    }

    if (max_row != pivot_row) {
        for (int j = 0; j <= N; j++) {
            double tmp = A[pivot_row][j];
            A[pivot_row][j] = A[max_row][j];
            A[max_row][j] = tmp;
        }
        printf("Swapped row %d and row %d.\n\n", pivot_row + 1, max_row + 1);
        print_matrix(A);
    }

    return 1;
}

```

```c
// ガウス・ジョルダン法の実装
int gauss_jordan(Matrix A) {
    for (int i = 0; i < N; i++) {
        if (!partial_pivot(A, i)) {
            return 0;
        }

        double pivot = A[i][i];
        for (int j = i; j <= N; j++) {
            A[i][j] /= pivot;
        }

        for (int k = 0; k < N; k++) {
            if (k == i) {
                continue;
            }
            double factor = A[k][i];
            for (int j = i; j <= N; j++) {
                A[k][j] -= factor * A[i][j];
            }
        }

        print_matrix(A);
    }
    return 1;
}

// 解の表示
void print_solutions(const Matrix A) {
    printf("--------- Solutions ---------\n");
    for (int i = 0; i < N; i++) {
        printf("x_%d = %8.3f\n", i + 1, A[i][N]);
    }
}

// メイン関数
int main(void) {
    const char* filename = "mat_Ab.txt";
    FILE* ifp = fopen(filename, "rt");
    if (!ifp) {
        fprintf(stderr, "Can't open file %s\n", filename);
        return 1;
    }

    Matrix A = {0.0}; //
    if (!scan_mat(ifp, A)) {
        fclose(ifp);
        return 1;
    }

    fclose(ifp);

    print_matrix(A);

    if (gauss_jordan(A)) {
        print_solutions(A);
    }

    return 0;
}
```