```c
// 6_2.cpp
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const int MAX_SIZE = 7;      // 行列の最大サイズ
const double EPS = 1e-6;     // ゼロ判定用のしきい値
const int STR_LEN = 200;     // 読み取り用バッファサイズ

typedef double Matrix[MAX_SIZE][MAX_SIZE * 2];  // A | I の拡大行列

// 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
    int row = 0;

    while (fgets(line, STR_LEN, ifp) != NULL && row < MAX_SIZE) {
        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 < MAX_SIZE; j++) {
            A[row][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。
        }
        row++;
    }

    return row;  // 実際の行列サイズを返す
}

// サンプルプログラムの行列を読み込む関数の解答。このプログラムでは未使用。
int scan_mat_sample(FILE* ifp, Matrix A) {
    int col[MAX_SIZE] = {0};

    for (int i = 0; i < MAX_SIZE + 1; i++) {
        char line[STR_LEN] = {0};
        int str_end = 0;

        if (fgets(line, STR_LEN, ifp) == NULL) {
            for (int k = 0; k < i; k++) {
                if (i != col[k]) {
                    printf("In row %d, number of columns is inconsistent.\n", k + 1);
                    return 0;
                }
            }
            return i;  // 行数を返す
        }

        for (int z = 0; z < STR_LEN + 1; z++) {
            if (line[z] == '\0' || line[z] == '\n') {
                str_end = z;
                break;
            }
        }

        char* ptr = line;
        for (int j = 0; j < MAX_SIZE + 1; j++) {
            A[i][j] = strtod(ptr, &ptr);
            if (ptr == &line[str_end]) {
                col[i] = j;
                break;
            }
            ++ptr;
        }
    }

    return 0;
}

// A に単位行列 I を右に結合
void make_combining_I_matrix(Matrix A, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = size; j < size * 2; j++) {
            A[i][j] = (i == j - size) ? 1.0 : 0.0;
        }
    }
}
```

```c

// 行列の表示
void print_matrix(const Matrix A, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size * 2; j++) {
            if (j == size) {
                printf(" |");
            }
            printf("%8.3f", A[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

// 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 size, int row) {
    int max_row = row;
    double max_val = fabs(A[row][row]);

    for (int i = row + 1; i < size; i++) {
        if (fabs(A[i][row]) > max_val) {
            max_val = fabs(A[i][row]);
            max_row = i;
        }
    }

    if (max_val < EPS) {
        printf("Error: Singular matrix.\n");
        return 0;
    }

    if (max_row != row) {
        for (int j = 0; j < size * 2; j++) {
            double tmp = A[row][j];
            A[row][j] = A[max_row][j];
            A[max_row][j] = tmp;
        }
        printf("Swapped row %d and row %d.\n\n", row + 1, max_row + 1);
        print_matrix(A, size);
    }

    return 1;
}

// 全要素がゼロの行があるか確認
int check_zero_row(const Matrix A, int size) {
    for (int i = 0; i < size; i++) {
        double sum = 0;
        for (int j = 0; j < size; j++) {
            sum += fabs(A[i][j]);
        }
        if (sum < EPS) {
            return 0;
        }
    }
    return 1;
}

// ガウス・ジョルダン法で逆行列を求める
int gauss_jordan(Matrix A, int size) {
    for (int i = 0; i < size; i++) {
        if (!partial_pivot(A, size, i)) {
            return 0;
        }

        double pivot = A[i][i];
        for (int j = 0; j < size * 2; j++) {
            A[i][j] /= pivot;
        }

        for (int k = 0; k < size; k++) {
            if (k == i) {
                continue;
            }
            double factor = A[k][i];
            for (int j = 0; j < size * 2; j++) {
```

```
                A[k][j] -= factor * A[i][j];
            }
        }

        print_matrix(A, size);

        if (!check_zero_row(A, size)) {
            printf("Matrix A is singular (rank deficient).\n");
            return 0;
        }
    }

    return 1;
}

// 逆行列の出力（右側だけ）
void print_inv_matrix(const Matrix A, int size) {
    printf("-------- Inverse Matrix --------\n");
    for (int i = 0; i < size; i++) {
        for (int j = size; j < size * 2; j++) {
            printf("%8.3f", A[i][j]);
        }
        printf("\n");
    }
}

int main(void) {
    const char* const ifile = "mat_A_rand.txt";
    FILE* ifp = fopen(ifile, "rt");

    if (!ifp) {
        fprintf(stderr, "Can't open file %s\n", ifile);
        return 1;
    }

    Matrix A = {0.0}; //初期化
    int size = scan_mat_sample(ifp, A);
    fclose(ifp);

    if (size > 0) {
        make_combining_I_matrix(A, size);
        print_matrix(A, size);

        if (gauss_jordan(A, size)) {
            print_inv_matrix(A, size);
        }
    } else {
        fprintf(stderr, "Failed to load matrix.\n");
    }

    return 0;
}
```