```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define ROW     3  /* 行の要素数 */
#define COLUMN  4  /* 列の要素数 */

/* 行列の領域確保 */
double **dmatrix(int nr1, int nr2, int nl1, int nl2);
/* 行列の領域解放 */
void free_dmatrix(double **a, int nr1, int nr2, int nl1, int nl2);
double *dvector(int i, int j);  /* ベクトル領域の確保 */
void free_dvector(double *a, int i1); /* 領域の解放 */

/* 比較関数 */
int double_comp(const void *s1, const void *s2);
/* 1 ノルムの計算 a[m1...m2][n1...n2] */
double matrix_norm1(double **a, int m1, int m2, int n1, int n2);
/* 最大値ノルムの計算 a[m1...m2][n1...n2] */
double matrix_norm_max(double **a, int m1, int m2, int n1, int n2);

int main(void)
{
        int i, j;
        double **a;

        a = dmatrix(1, ROW, 1, COLUMN);  /* 行列 a[1...ROW][1...COLUMN] */
        /* 行列の定義 */
        printf("A=\n");
        for(i = 1; i <= ROW; i++){
                for(j = 1; j <= COLUMN; j++){
                        a[i][j] = 2.0 * (i + j) * pow(-1.0,j);
                        printf("%f\t", a[i][j]);
                }
                printf("\n");
        }

        printf("A の 1 ノルムは%f\n", matrix_norm1(a, 1, ROW, 1, COLUMN));
        printf("A の最大値ノルムは%f\n", matrix_norm_max(a, 1, ROW, 1, COLUMN));

        /* 行列領域の解放 */
        free_dmatrix(a, 1, ROW, 1, COLUMN);

        return 0;
}

double **dmatrix(int nr1, int nr2, int nl1, int nl2)
{
        int i, nrow, ncol;
        double **a;

        nrow = nr2 - nr1 + 1; /* 行の数 */
        ncol = nl2 - nl1 + 1; /* 列の数 */

        /* 行の確保 */
        if ((a = malloc(nrow * sizeof(double *))) == NULL)
        {
                printf("メモリが確保できません (行列 a)\n");
                exit(1);
        }
        a = a - nr1; /* 行をずらす */

        /* 列の確保 */
        for (i = nr1; i <= nr2; i++)
                a[i] = malloc(ncol * sizeof(double));
        for (i = nr1; i <= nr2; i++)
                a[i] = a[i] - nl1; /* 列をずらす */

        return (a);
}

void free_dmatrix(double **a, int nr1, int nr2, int nl1, int nl2)
{
        int i;

        /* メモリの解放 */
        for (i = nr1; i <= nr2; i++)
                free((void *)(a[i] + nl1));
        free((void *)(a + nr1));
}

double *dvector(int i, int j) /* a[i]〜a[j] の領域を確保 */
{
        double *a;

        if ((a = malloc(((j - i + 1) * sizeof(double)))) == NULL)
        {
                printf("メモリが確保できません (from dvector) \n");
                exit(1);
        }

        return (a - i);
}

void free_dvector(double *a, int i)
{
        free((void *)(a + i)); /* (void *) 型へのキャストが必要 */
}

/* 比較関数 (昇順) */
int double_comp(const void *s1, const void *s2)
{
        const double a1 = *((double *)s1); /* (double *) へキャスト */
        const double a2 = *((double *)s2); /* (double *) へキャスト */
        if (a1 < a2)
        {
                return -1;
        }
        else if (a1 == a2)
        {
                return 0;
        }
        else
        {
                return 1;
        }
}

/* 1 ノルムの計算 */
double matrix_norm1(double **a, int m1, int m2, int n1, int n2)
{
        int i, j, tmp;
        double *work, norm;
        work = dvector(n1, n2);  /* ベクトル work[n1...n2] */

        /* 列和の計算 */
        for(j = n1; j <= n2; j++){
                work[j] = 0.0;
                for(i = m1; i <= m2; i++){
                        work[j] += fabs(a[i][j]);
                }
        }

        tmp = n2 - n1 + 1;  /* 列数の計算 */
        /* 並べ換え:先頭アドレスが n1 だけずれていることに注意 */
        qsort(work + n1, tmp, sizeof(work[0]), double_comp);

        norm = work[n2];
        free_dvector(work, n1);

        return norm;
}

/* 最大値ノルムの計算 */
double matrix_norm_max(double **a, int m1, int m2, int n1, int n2)
{
        int i, j, tmp;
        double *work, norm;
        work = dvector(m1, m2); /* ベクトル work[m1...m2] */

        /* 行和の計算 */
        for(i = m1; i <= m2; i++){
                work[i] = 0.0;
                for(j = n1; j <= n2; j++){
                        work[i] += fabs(a[i][j]);
                }
        }

        tmp = m2 - m1 + 1;  /* 行数の計算 */

        /* 並べ換え:先頭アドレスが m1だけずれていることに注意 */
        qsort(work + m1, tmp, sizeof(work[0]), double_comp);

        norm = work[m2];
        free_dvector(work, m1);

        return norm;
}
```