```
/* *************************************************
   This file contains the routines for reading the
   TSPLIB95 files into an adjacency matrix.
   ************************************************* */


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <strings.h>

#define _USE_MATH_DEFINES
#include <math.h>

#include "tsplib95.h"

#define NAME_FIELD "NAME"
#define DIM_FIELD "DIMENSION"
#define EW_TYPE_FIELD "EDGE_WEIGHT_TYPE"
#define EW_FORMAT_FIELD "EDGE_WEIGHT_FORMAT"
#define EW_SECTION_FIELD "EDGE_WEIGHT_SECTION"

#define EW_TYPE_EXPLICIT "EXPLICIT"
#define EW_FORMAT_LDR "LOWER_DIAG_ROW"

#define EW_EOF "EOF"

#define RRR 6378.388

/* Parse a TSPLIB95 file and return the adjacency matrix. */
void parse_matrix_from_file(struct tsp_matrix *matrix, FILE *file) {
        char line[1000];

        char *name, *ew_type, *ew_format;

        name = calloc( 100, sizeof(&name));
        ew_type = calloc( 100, sizeof(&ew_type));
        ew_format = calloc( 100, sizeof(&ew_format));

        while( fgets( line, sizeof(line), file) != NULL ) {
                if( strncmp(line, NAME_FIELD, strlen(NAME_FIELD)) == 0 ) {
                        sscanf(line, "NAME: %s", matrix->name);
                }
                else if( strncmp( line, DIM_FIELD, strlen(DIM_FIELD)) == 0 ) {
                        sscanf(line, "DIMENSION: %d", &matrix->n);
                        /* allocate the matrix */
                        matrix->data = calloc( matrix->n*matrix->n, sizeof(*matrix->data));
                }
                else if( strncmp( line, EW_TYPE_FIELD, strlen(EW_TYPE_FIELD)) == 0 ) {
                        sscanf(line, "EDGE_WEIGHT_TYPE: %s", ew_type);
                }
                else if( strncmp( line, EW_FORMAT_FIELD, strlen(EW_FORMAT_FIELD)) == 0 ) {
                        sscanf(line, "EDGE_WEIGHT_FORMAT: %s", ew_format);
                }
                else if( strncmp( line, EW_SECTION_FIELD, strlen(EW_SECTION_FIELD)) == 0 ||
                         strncmp( line, "NODE_COORD_SECTION", strlen("NODE_COORD_SECTION")) == 0 ) {
                        /* parse the data */
                        if( strncmp(ew_type, EW_TYPE_EXPLICIT, strlen(EW_TYPE_EXPLICIT)) == 0 &&
                            strncmp(ew_format, EW_FORMAT_LDR, strlen(EW_FORMAT_LDR)) == 0 ) {
                                parse_explicit_lowerdiagrow(matrix, file);
                        }
                        else if( strncmp(ew_type, "GEO", 3) == 0 &&
                                 strncmp(ew_format, "FUNCTION", 8) == 0 ) {
                                parse_geo_function(matrix, file);
                        }
                        else {
                                printf("PARSE ERROR: Don't know how to read %s, %s.\n", ew_type,
ew_format);
                                exit(1);
                        }
                }
        }

        /*
        printf("The file name is: %s\n", matrix->name);
        printf("The dimension is: %d\n", matrix->n);
        printf("The edge weight type is: %s\n", ew_type);
        printf("The edge weight format is: %s\n", ew_format);
        */

        return;
}

/* PARSE: GEO, FUNCTION */
void parse_geo_function(struct tsp_matrix *matrix, FILE *file) {

        char line[1000];
        int token_num = 0;
        int row_num = 0;
        int i, j;

        double *x, *y, *latitude, *longitude;
        double deg, min, q1, q2, q3;

        x = calloc( matrix->n, sizeof(*x));
        y = calloc( matrix->n, sizeof(*y));
        latitude = calloc( matrix->n, sizeof(*latitude));
        longitude = calloc( matrix->n, sizeof(*longitude));

        i = 0;
        while( fgets( line, sizeof(line), file) != NULL ) {

                if( strncmp(line, EW_EOF, strlen(EW_EOF)) == 0 ) break;

                char *token = strtok(line, " ");

                j = 0;
                while( token != NULL && strcmp(token,"\n") != 0 ) {
                        if( strlen(token) == 0 ) {
                                token = strtok(NULL, " ");
                                continue;
                        }


                        if( j == 1 ) {
                                x[i] = atof(token); // x coordinate
                        }

                        if( j == 2 ) {
                                y[i] = atof(token); // y coordinate
                        }

                        token = strtok( NULL, " ");
                        j++;
                }
                i++;
        }

        for( i = 0; i < matrix->n; i++ ) {
                deg = (int) ( x[i] ); // degrees (integer part)
                min = x[i] - deg;   // minutes (decimal part)
                latitude[i] = 3.141592 * ( deg + 5.0 * min / 3.0 ) / 180.0;

                deg = (int) ( y[i] ); // degrees (integer part)
                min = y[i] - deg;   // minutes (decimal part)
                longitude[i] = 3.141592 * ( deg + 5.0 * min / 3.0 ) / 180.0;
        }

        for( i = 0; i < matrix->n; i++ ) {
                for( j = 0; j < matrix->n; j++ ) {
                        if( i == j ) continue; // diagonals are 0

                        q1 = cos( longitude[i] - longitude[j] );
                        q2 = cos( latitude[i] - latitude[j] );
                        q3 = cos( latitude[i] + latitude[j] );

                        matrix->data[i*matrix->n + j] = (int) ( RRR * acos( 0.5*( (1.0+q1)*q2 - (1.0-q1)*q3 ) ) + 1.0 );
                }
        }


        free(x);
        free(y);
        free(latitude);
        free(longitude);

}

void parse_explicit_lowerdiagrow(struct tsp_matrix *matrix, FILE *file) {

        char line[1000];
        int token_num = 0;
        int row_num = 0;
        int i, j;

        while( fgets( line, sizeof(line), file) != NULL ) {

                if( strncmp(line, EW_EOF, strlen(EW_EOF)) == 0 ) break;

                char *token = strtok(line, " ");
                while( token != NULL && strlen(token) != 0 && strcmp(token,"\n") != 0 ) {

                        /* printf("token: %s, %ld\n", token, strlen(token)); */

                        matrix->data[row_num*matrix->n + token_num] = atoi(token);

                        if( token_num == row_num ) {
                                token_num = 0;
                                row_num++;
                        } else {
                                token_num++;
                        }

                        token = strtok( NULL, " ");
                }
        }

        /* ok, now copy the lower diagonal to the upper diagonal */
        for( i = 0; i < matrix->n; i++ ) {
                for( j = i+1; j < matrix->n; j++ ) {
                        matrix->data[i*matrix->n + j] = matrix->data[j*matrix->n + i];
                }
        }
}


void print_matrix(struct tsp_matrix *m)
{
        int i;
        int count = m->n*m->n;
        int *p = m->data;

        for (i = 1; i <= count; i++) {
                printf("%4d", *p);
                if (i % m->n == 0) printf("\n");
                else printf(" ");
                p++;
        }
}
```