```c
#include "dataManager.h"

// FILE READERS

/* readStationData: parse csv file for stations
 * \param   filename        path and filename of csv file for stations
 * \return  stationsHead    list of stations header
 */
Station * readStationData(char *filename){
    char line[1024];
    char *token;
    char *separators = ",";
    int lineNumber = 0;
    int fieldCounter = 0;

    Station * stationsHead = NULL;

    //open file for reading
    FILE *fileTwo = fopen( filename, "r" );
    if ( fileTwo == 0 ) {
        printf( "Error - Could not open stations file: %s\n", filename );
        exit(EXIT_FAILURE);
    }
    else {
        while(fgets(line, sizeof line, fileTwo) != NULL){
            // keep line count for convenience
            lineNumber++;
            // Split the line into parts
            token = strtok(line, separators);
            // make sure field counter is 0
            fieldCounter = 0;
            // Allocation of memory
            Station* station = (Station*)malloc(sizeof(Station));
            // Skip first line with headers
            if (lineNumber != 1) {
                // cycle through fields
                while (token != NULL) {
                    // printf("Field: %d\n", fieldCounter);
                    switch (fieldCounter) {
                        case 0:  station->id = atoi(token);
                            break;
                        case 1:  strcpy(station->name, token);
                            break;
                        case 2:  strcpy(station->full_name, token);
                            break;
                        case 3:  strcpy(station->municipal, token);
                            break;
                        case 4:  station->latitude = atof(token);
                            break;
                        case 5:  station->longitude = atof(token);
                            break;
                        case 6:
                            if (strcmp(token, "Existing") == 0) {
                                station->status = EXISTING;
                            } else {
                                station->status = REMOVED;
                            }
                            break;
                        default: break;
                    }
                    // printf ("%s\n",token);
                    fieldCounter++;
                    token = strtok (NULL, separators);
                }
                // add new trip to linked list
                station->next = stationsHead;
                stationsHead = station;
            }
        }
        fclose(fileTwo);
    }
    return stationsHead;
}

/* readTripsData: parse csv file for trips
 * \param   filename        path and filename of csv file for trips
 * \return  tripsHead      list of trips header
 */
Trip * readTripsData(char *filename){
    char line[1024];
    char *token;
    char *separators = ", /:\n";
    int lineNumber = 0;
    int fieldCounter = 0;

    Trip * tripsHead = NULL;

    //open file for reading
    FILE *fileOne = fopen( filename, "r" );
    if ( fileOne == 0 ) {
        printf( "Error - Could not open trips file: %s\n", filename );
        exit(EXIT_FAILURE);
    }
    else {
        // read each line
        while(fgets(line, sizeof line, fileOne) != NULL){
            // keep line count for convenience
            lineNumber++;
            // Split the line into parts
            token = strtok(line, separators);
            // make sure field counter is 0
            fieldCounter = 0;
            // Allocation of memory
            Trip* trip = (Trip*)malloc(sizeof(Trip));
            // cycle through fields
            while (token != NULL) {

                //printf("%d %s\n", fieldCounter, token);
                switch (fieldCounter) {
                    case 0:  trip->id = atoi(token);                break;
                    case 1:  trip->duration = atoi(token);          break;
                    case 2:  trip->start.month = atoi(token);       break;
                    case 3:  trip->start.day = atoi(token);         break;
                    case 4:  trip->start.year = atoi(token);        break;
                    case 5:  trip->start.hour = atoi(token);        break;
                    case 6:  trip->start.minute = atoi(token);      break;
                    case 7:  /*"seconds field" to be ignored*/      break;
                    case 8:  trip->id_start_station = atoi(token);  break;
                    case 9:  trip->end.month = atoi(token);         break;
                    case 10: trip->end.day = atoi(token);           break;
                    case 11: trip->end.year = atoi(token);          break;
                    case 12: trip->end.hour = atoi(token);          break;
                    case 13: trip->end.minute = atoi(token);        break;
                    case 14: /*"seconds field" to be ignored*/      break;
                    case 15: trip->id_final_station = atoi(token);  break;
                    case 16:
                        // handle missing bike id on line 609
                        if (strlen(token)-1 > 8) {
                            if (strcmp(token, "Registered") !=0) {
                                trip->type = REGISTERED;
                            } else {
                                trip->type = CASUAL;
                            }
                            // add one to field counter so we skip
                            // this field
                            fieldCounter++;
                            break;
                        }
                        strcpy(trip->bike, token);
                        break;
                    case 17:
                        if (strcmp(token, "Registered") == 0) {
                            trip->type = REGISTERED;
                        } else {
                            trip->type = CASUAL;
                        }
                        break;
                    case 18:
                        trip->year_birthday = atoi(token);
                        break;
                    case 19:
                        if (token[0] == 'M') {
                            trip->gender = MALE;
                        } else if (token[0] == 'F') {
                            trip->gender = FEMALE;
                        } else {
                            trip->gender = 0;
                        }
                        break;
                    default: break;
                }
                // printf ("%s\n",token);
                fieldCounter++;
                token = strtok (NULL, separators);
            }
            // add new trip to linked list
            trip->next = tripsHead;
            tripsHead = trip;
        }
        fclose(fileOne);
    }
    return tripsHead;
}

// LIST CREATORS

/* createRoutesList: returns the list of routes, ordered descendant
 * \param   tripList              the head of the trips list
 *                                (can be filtered)
 * \param   allStations           the head of all stations list
 * \param   selected_station_id   the ID of the selected station
 * \return  routes                the head of the routes list
 */
Route * createRoutesList(Trip * tripList, Station * allStations,
                         int selected_station_id) {
    Route * routes = NULL;
    Station * auxStations = allStations;

    char selected_station_name[7];
    strcpy(selected_station_name,
           getStationNameById(selected_station_id, allStations));

    // foreach station
    while (auxStations != NULL) {

        // initialize route counters
        int tripsIn = 0;
        int tripsOut = 0;

        // go through the trips list
        Trip * auxTrips = tripList;

        while (auxTrips != NULL) {

            if (auxStations->id != selected_station_id) {
                // count trips from current station to selected station
                if (auxTrips->id_start_station == auxStations->id) {
                    tripsOut++;
                }
                // count trips from selected station to current station
                else if (auxTrips->id_final_station == auxStations->id) {
                    tripsIn++;
                }
                // make sure we only update one counter if the trip is
                // from and to the same station
            } else {
                tripsOut++;
            }
            auxTrips = auxTrips->next;
        }
        if (tripsOut > 0) {
            // Create route: from current station to selected station
            Route * routeIn = malloc(sizeof(Route));

            routeIn->total = tripsOut;
            routeIn->id_start_station = auxStations->id;
            strcpy(routeIn->name_start_station, auxStations->name);
            routeIn->id_final_station = selected_station_id;
            strcpy(routeIn->name_final_station, selected_station_name);

            sortedInsert(&routes, routeIn);
        }

        if (tripsIn > 0) {
            // Create route: from selected stations to current station
            Route * routeOut = malloc(sizeof(Route));

            routeOut->total = tripsIn;
            routeOut->id_final_station = auxStations->id;
            strcpy(routeOut->name_final_station, auxStations->name);
            routeOut->id_start_station = selected_station_id;
            strcpy(routeOut->name_start_station, selected_station_name);

            sortedInsert(&routes, routeOut);
        }
        auxStations = auxStations->next;
    }
    return routes;
}

/* countBikes: returns list of stations with all the max/min/avg populated
 * \param   tripList              the head of the trips list
 *                                (can be filtered)
 * \param   stationsList          the head of stations list
 * \param   filtered_hour_start   the start hour for the selectTripsByTime
 * \param   filtered_hour_end     the end hour for the selectTripsByTime
 * \return  stationsList          the head of the stations list,
 *                                with all calculated data added
 */
Station * countBikes(Trip *tripsList, Station *stationsList,
                     int filter_hour_start, int filter_hour_end) {
    struct Station * auxStations = stationsList;


    while (auxStations != NULL) {

        Trip *trips = selectTripsByIdStation(tripsList, auxStations->id);

        // initialize counters and hours
        int tripsCount = 0;
        int inTotal = 0;
        int outTotal = 0;

        int counterIn[24] = {0};
        int counterOut[24] = {0};

        if (trips != NULL) {
            while (trips != NULL) {
                tripsCount++;
                if (trips->id_start_station == auxStations->id) {
                    counterOut[trips->start.hour]++;
                }
                if (trips->id_final_station == auxStations->id) {
                    counterIn[trips->end.hour]++;
                }

                trips = trips->next;
            }
        }
        int maxIn = counterIn[0];
        int minIn = counterIn[0];
        int maxOut = counterOut[0];
        int minOut = counterOut[0];

        int start = 0;
        int end = 24;
        if (filter_hour_start != -1 && filter_hour_end != -1) {

            start = filter_hour_start;
            end = filter_hour_end;

            maxIn = counterIn[filter_hour_start];
            minIn = counterIn[filter_hour_start];
            maxOut = counterOut[filter_hour_start];
            minOut = counterOut[filter_hour_start];
        }

        // find max and min within selected time range
        if (start < end) {
            for (int i = start; i<end; i++){
                if (maxIn < counterIn[i]) maxIn = counterIn[i];
                if (maxOut < counterOut[i]) maxOut = counterOut[i];
                if (minIn > counterIn[i]) minIn = counterIn[i];
                if (minOut > counterOut[i]) minOut = counterOut[i];

                inTotal += counterIn[i];
                outTotal += counterOut[i];
            }

            // handle scenario: hour start is < then hour end
            // (i.e. 22 to 4)
        } else if (start > end) {
            for (int i = start; i<24; i++) {
                if (maxIn < counterIn[i]) maxIn = counterIn[i];
                if (maxOut < counterOut[i]) maxOut = counterOut[i];
                if (minIn > counterIn[i]) minIn = counterIn[i];
                if (minOut > counterOut[i]) minOut = counterOut[i];

                inTotal += counterIn[i];
                outTotal += counterOut[i];
            }
            for (int i = 0; i<end; i++) {
                if (maxIn < counterIn[i]) maxIn = counterIn[i];
                if (maxOut < counterOut[i]) maxOut = counterOut[i];
                if (minIn > counterIn[i]) minIn = counterIn[i];
                if (minOut > counterOut[i]) minOut = counterOut[i];

                inTotal += counterIn[i];
                outTotal += counterOut[i];
            }
        }
        else {
            if (maxIn < counterIn[start]) maxIn = counterIn[start];
            if (maxOut < counterOut[start]) maxOut = counterOut[start];
            if (minIn > counterIn[start]) minIn = counterIn[start];
            if (minOut > counterOut[start]) minOut = counterOut[start];

            inTotal += counterIn[start];
            outTotal += counterOut[start];
        }

        // save counters data to Stations list
        auxStations->max_bikesIn    = maxIn;
        auxStations->max_bikesOut   = maxOut;
        auxStations->min_bikesIn    = minIn;
        auxStations->min_bikesOut   = minOut;

        // calculate average
        if (start < end) {
            auxStations->avg_bikesIn    = inTotal/(end-start);
            auxStations->avg_bikesOut   = outTotal/(end-start);
        } else if (start == end) {
            auxStations->avg_bikesIn    = inTotal;
            auxStations->avg_bikesOut   = outTotal;
        } if (start > end) {
            auxStations->avg_bikesIn    = inTotal/((24-start)+end);
            auxStations->avg_bikesOut    = outTotal/((24-start)+end);
        }
        auxStations = auxStations->next;
    }
    return stationsList;
}


// LIST FILTERS

/* selectTripsByTime: returns list of trips between hour star
 *                    and hour end
 * \param   sourceListHead        the head of the trips list
 *                                (can be filtered)
 * \param   hour_start            the start hour for the selectTripsByTime
filter
 * \param   hour_end              the end hour for the
 *                                selectTripsByTime filter
 * \return  filteredTripsHead     the head of the trips list, filtered
 */
Trip* selectTripsByTime(Trip * sourceListHead, int hour_start,
                        int hour_end) {
    Trip *aux = sourceListHead;
    Trip *filteredTripsHead = NULL;
    while (aux != NULL) {

        // Only save the item to the list if the start hour
        // and end hour are within the parameters
        bool shouldSave = false;

        // if hour start < hour end, then check if trip is between
        // time span. if user enters trips from 8 to 9, we take all
        // trips between 8:00 and 8:59
        if (hour_start < hour_end) {
            if (    (aux->start.hour >= hour_start) &&
                    (aux->start.hour < hour_end)    &&
                    (aux->end.hour >= hour_start)   &&
                    (aux->end.hour < hour_end) )    {
                shouldSave = true;
            }
        }
        // handle scenario: hour start > hour end
        // if user enters trips from 18 to 17,
        // we take all trips between 18:00 and 23:59,
        // and all trips between 0:00 and 17:59
        else if (hour_start > hour_end) {
            if ( ( (aux->start.hour >= hour_start)  &&
                    (aux->end.hour <= 23) )         ||
                    ((aux->start.hour >= 0)         &&
                    (aux->end.hour < hour_end)) )   {
                // get all trips between hour_start and 23:59.
                // get all trips between 0 and hour_end.
                shouldSave = true;
            }
        }
        // handle scenario: hour start = hour end
        // if user enters trips from 16 to 16, we take all trips
        // between 16:00 and 16:59
        else {
            if (    (aux->start.hour == hour_start) &&
                    (aux->end.hour == hour_end) )   {
                shouldSave = true;
            }
        }
        // save item if it should
        if (shouldSave) {
            filteredTripsHead = copyTripToList(filteredTripsHead, aux);
        }

        aux = aux->next;
    }
    return filteredTripsHead;
}

/* selectTripsByDuration: returns list of trips given max duration
 * \param   sourceListHead        the head of the trips list
 *                                (can be filtered)
 * \param   duration              the maximum duration of a trip in
 *                                seconds
 * \return  filteredTripsHead     the head of the trips list, filtered
 */
Trip* selectTripsByDuration(Trip * sourceListHead, int duration) {
    struct Trip *aux = sourceListHead;
    struct Trip *filteredTripsHead = NULL;
    while (aux != NULL) {
        if (aux->duration <= duration) {

            filteredTripsHead = copyTripToList(filteredTripsHead, aux);
        }
        aux = aux->next;
    }
    return filteredTripsHead;
}

/* selectTripsByDay: returns list of trips given day of week
 * \param   sourceListHead        the head of the trips list
 *                              (can be filtered)
 * \param   selectedDay          an int representing the day of the
 *                              week (1 monday..7 sunday)
 * \return  filteredTripsHead    the head of the trips list, filtered
 */
Trip* selectTripsByDay(Trip * sourceListHead, int selectedDay){

    struct Trip *aux = sourceListHead;
    struct Trip *filteredTripsHead = NULL;

    if (selectedDay == 7) {
        selectedDay = 0;
    }

    while (aux != NULL) {

        // Check if day of the current trip is == to the selected day
        if ((calculateWeekDateFromDate(aux->start.year, aux->start.month,
            aux->start.day) == selectedDay) ||
            (calculateWeekDateFromDate(aux->end.year, aux->end.month,
            aux->end.day) == selectedDay)){

            filteredTripsHead = copyTripToList(filteredTripsHead, aux);
        }
        aux = aux->next;
    }
    return filteredTripsHead;
}

/* selectTripsByIdStation: returns list of trips given a station ID
 * \param   sourceListHead       the head of the trips list
 *                              (can be filtered)
 * \param   id                   the station ID
 * \return  filteredTripsHead    the head of the trips list, filtered
 */
Trip* selectTripsByIdStation(Trip * sourceListHead, int id) {
    int counter=0;
    struct Trip *aux = sourceListHead;
    struct Trip *filteredTripsHead = NULL;
    while (aux != NULL) {

        if ((aux->id_final_station == id)||(aux->id_start_station == id)){
            filteredTripsHead = copyTripToList(filteredTripsHead, aux);
            counter++;
        }

        aux = aux->next;
    }
    return filteredTripsHead;
}

// HELPERS

/* copyTripToList: add element at top of Trip List
 * \param   filteredTripsHead    the head of the trips list to add
 *                              the element into
 * \param   aux                  the trip to be added to the list
 * \return  filteredTripsHead    the head of the trips list, with
 *                              the new element at the top
 */
Trip* copyTripToList(Trip * filteredTripsHead, Trip * aux) {

    Trip* trip = (Trip*)malloc(sizeof(Trip));

    trip->id = aux->id;
    strcpy(trip->bike, aux->bike);
    trip->duration = aux->duration;
    trip->end = aux->end;
    trip->start = aux->start;
    trip->gender = aux->gender;
    trip->id_final_station = aux->id_final_station;
    trip->id_start_station = aux->id_start_station;
    trip->type = aux->type;
    trip->year_birthday = aux->year_birthday;

    trip->next = filteredTripsHead;
    filteredTripsHead = trip;

    return filteredTripsHead;
}

/* calculateWeekDateFromDate: Calculate the day of the week this
*                             current trip was in
 * -Source:
 * stackoverflow.com/questions/6054016/c-program-to-find-day-of-week-given-date
 * \param   y       year (4 digits)
 * \param   m       month (2 digits)
 * \param   d       day (2 digits)
 * \return  weekday number 1 to 7 with 1 = monday and 7 = sunday
 */
int calculateWeekDateFromDate(int y, int m, int d) {
  int weekday=(d+=m<3 ? y-- : y - 2, 23*m/9 +d+ 4 + y/4- y/100 + y/400)%7;
  return weekday;
}

/* sortedInsert: insert a route in the correct order
 * \param   head_ref       the head of the list to add the route into
 * \param   new_node       the new route to add to the list
 */
void sortedInsert(Route** head_ref, Route* new_node) {
    Route* current;
    // handle scenario for the head end
    if (*head_ref == NULL || (*head_ref)->total <= new_node->total) {
        new_node->next = *head_ref;
        *head_ref = new_node;
    }
    else
    {
        // find the node before the point of insert
        current = *head_ref;
        while (current->next != NULL &&
               current->next->total > new_node->total) {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
    }
}

/* getStationNameById: Get a Station Name by its ID
 * \param   id                  the ID of the station to look for
 * \param   allStations         the head of all stations list
 * \return  auxStations->name   containing the station name,
 *                              or an empty string
 *                              if no station was found
 */
char * getStationNameById(int id, Station * allStations) {
    Station * auxStations = allStations;
    while (auxStations != NULL) {
        if (auxStations->id == id) {
            return auxStations->name;
        }
        auxStations = auxStations->next;
    }
    return "";
}


```