```c

/* Project created by Sara Marfella IST188316 on May 16, 2017 */

#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[MAX_SIZE];
    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);
```

```
67                  }
68                  // add new trip to linked list
69                  station->next = stationsHead;
70                  stationsHead = station;
71              }
72          }
73          fclose(fileTwo);
74      }
75      return stationsHead;
76  }
77
78  /* readTripsData: parse csv file for trips
79   * \param   filename        path and filename of csv file for trips
80   * \return  tripsHead     list of trips header
81   */
82  Trip * readTripsData(char *filename){
83      char line[MAX_SIZE];
84      char *token;
85      char *separators = ", /:\n";
86      int lineNumber = 0;
87      int fieldCounter = 0;
88
89      Trip * tripsHead = NULL;
90
91      //open file for reading
92      FILE *fileOne = fopen( filename, "r" );
93      if ( fileOne == 0 ) {
94          printf( "Error - Could not open trips file: %s\n", filename );
95          exit(EXIT_FAILURE);
96      }
97      else {
98          // read each line
99          while(fgets(line, sizeof line, fileOne) != NULL){
100             // keep line count for convenience
101             lineNumber++;
102             // Split the line into parts
103             token = strtok(line, separators);
104             // make sure field counter is 0
105             fieldCounter = 0;
106             // Allocation of memory
107             Trip* trip = (Trip*)malloc(sizeof(Trip));
108             // cycle through fields
109             while (token != NULL) {
110
111                 //printf("%d %s\n", fieldCounter, token);
112                 switch (fieldCounter) {
113                     case 0:  trip->id = atoi(token);                break;
114                     case 1:  trip->duration = atoi(token);          break;
115                     case 2:  trip->start.month = atoi(token);       break;
116                     case 3:  trip->start.day = atoi(token);         break;
117                     case 4:  trip->start.year = atoi(token);        break;
118                     case 5:  trip->start.hour = atoi(token);        break;
119                     case 6:  trip->start.minute = atoi(token);      break;
120                     case 7:  /*"seconds field" to be ignored*/      break;
121                     case 8:  trip->id_start_station = atoi(token);  break;
122                     case 9:  trip->end.month = atoi(token);         break;
123                     case 10: trip->end.day = atoi(token);           break;
124                     case 11: trip->end.year = atoi(token);          break;
125                     case 12: trip->end.hour = atoi(token);          break;
126                     case 13: trip->end.minute = atoi(token);        break;
127                     case 14: /*"seconds field" to be ignored*/      break;
128                     case 15: trip->id_final_station = atoi(token);  break;
129                     case 16:
130                         // handle missing bike id on line 609
131                         if (strlen(token)-1 > 8) {
132                             if (strcmp(token, "Registered") !=0) {
```

```
133                                 trip->type = REGISTERED;
134                             } else {
135                                 trip->type = CASUAL;
136                             }
137                             // add one to field counter so we skip
138                             // this field
139                             fieldCounter++;
140                             break;
141                         }
142                         strcpy(trip->bike, token);
143                         break;
144                     case 17:
145                         if (strcmp(token, "Registered") == 0) {
146                             trip->type = REGISTERED;
147                         } else {
148                             trip->type = CASUAL;
149                         }
150                         break;
151                     case 18:
152                         trip->year_birthday = atoi(token);
153                         break;
154                     case 19:
155                         if (token[0] == 'M') {
156                             trip->gender = MALE;
157                         } else if (token[0] == 'F') {
158                             trip->gender = FEMALE;
159                         } else {
160                             trip->gender = 0;
161                         }
162                         break;
163                     default: break;
164                 }
165                 // printf ("%s\n",token);
166                 fieldCounter++;
167                 token = strtok (NULL, separators);
168             }
169             // add new trip to linked list
170             trip->next = tripsHead;
171             tripsHead = trip;
172         }
173         fclose(fileOne);
174     }
175     return tripsHead;
176 }
177
178 // LIST CREATORS
179
180 /* createRoutesList: returns the list of routes, ordered descendant
181  * \param   tripList              the head of the trips list
182  *                                (can be filtered)
183  * \param   allStations           the head of all stations list
184  * \param   selected_station_id   the ID of the selected station
185  * \return  routes                the head of the routes list
186  */
187 Route * createRoutesList(Trip * tripList, Station * allStations,
188                          int selected_station_id) {
189     Route * routes = NULL;
190     Station * auxStations = allStations;
191
192     char selected_station_name[ID_SIZE];
193     strcpy(selected_station_name,
194            getStationNameById(selected_station_id, allStations));
195
196     // foreach station
197     while (auxStations != NULL) {
198
```

```
199         // initialize route counters
200         int tripsIn = 0;
201         int tripsOut = 0;
202 
203         // go through the trips list
204         Trip * auxTrips = tripList;
205 
206         while (auxTrips != NULL) {
207 
208             if (auxStations->id != selected_station_id) {
209                 // count trips from current station to selected station
210                 if (auxTrips->id_start_station == auxStations->id) {
211                     tripsOut++;
212                 }
213                 // count trips from selected station to current station
214                 else if (auxTrips->id_final_station == auxStations->id) {
215                     tripsIn++;
216                 }
217                 // make sure we only update one counter if the trip is
218                 // from and to the same station
219             } else {
220                 tripsOut++;
221             }
222             auxTrips = auxTrips->next;
223         }
224         if (tripsOut > 0) {
225             // Create route: from current station to selected station
226             Route * routeIn = malloc(sizeof(Route));
227 
228             routeIn->total = tripsOut;
229             routeIn->id_start_station = auxStations->id;
230             strcpy(routeIn->name_start_station, auxStations->name);
231             routeIn->id_final_station = selected_station_id;
232             strcpy(routeIn->name_final_station, selected_station_name);
233 
234             sortedInsert(&routes, routeIn);
235         }
236 
237         if (tripsIn > 0) {
238             // Create route: from selected stations to current station
239             Route * routeOut = malloc(sizeof(Route));
240 
241             routeOut->total = tripsIn;
242             routeOut->id_final_station = auxStations->id;
243             strcpy(routeOut->name_final_station, auxStations->name);
244             routeOut->id_start_station = selected_station_id;
245             strcpy(routeOut->name_start_station, selected_station_name);
246 
247             sortedInsert(&routes, routeOut);
248         }
249         auxStations = auxStations->next;
250     }
251     return routes;
252 }
253 
254 /* countBikes: returns list of stations with all the max/min/avg populated
255  * \param   tripList              the head of the trips list
256  *                                (can be filtered)
257  * \param   stationsList          the head of stations list
258  * \param   filtered_hour_start   the start hour for the selectTripsByTime
259  * \param   filtered_hour_end     the end hour for the selectTripsByTime
260  * \return  stationsList          the head of the stations list,
261  *                                with all calculated data added
262  */
263 Station * countBikes(Trip *tripsList, Station *stationsList,
264                      int filter_hour_start, int filter_hour_end) {
```

```
265     struct Station * auxStations = stationsList;
266
267
268     while (auxStations != NULL) {
269
270         Trip *trips = selectTripsByIdStation(tripsList, auxStations->id);
271
272         // initialize counters and hours
273         int tripsCount = 0;
274         int inTotal = 0;
275         int outTotal = 0;
276
277         int counterIn[24] = {0};
278         int counterOut[24] = {0};
279
280         if (trips != NULL) {
281             while (trips != NULL) {
282                 tripsCount++;
283                 if (trips->id_start_station == auxStations->id) {
284                     counterOut[trips->start.hour]++;
285                 }
286                 if (trips->id_final_station == auxStations->id) {
287                     counterIn[trips->end.hour]++;
288                 }
289
290                 trips = trips->next;
291             }
292         }
293         int maxIn = counterIn[0];
294         int minIn = counterIn[0];
295         int maxOut = counterOut[0];
296         int minOut = counterOut[0];
297
298         int start = 0;
299         int end = 24;
300         if (filter_hour_start != -1 && filter_hour_end != -1) {
301
302             start = filter_hour_start;
303             end = filter_hour_end;
304
305             maxIn = counterIn[filter_hour_start];
306             minIn = counterIn[filter_hour_start];
307             maxOut = counterOut[filter_hour_start];
308             minOut = counterOut[filter_hour_start];
309         }
310
311         // find max and min within selected time range
312         if (start < end) {
313             for (int i = start; i<end; i++){
314                 if (maxIn < counterIn[i]) maxIn = counterIn[i];
315                 if (maxOut < counterOut[i]) maxOut = counterOut[i];
316                 if (minIn > counterIn[i]) minIn = counterIn[i];
317                 if (minOut > counterOut[i]) minOut = counterOut[i];
318
319                 inTotal += counterIn[i];
320                 outTotal += counterOut[i];
321             }
322
323         // handle scenario: hour start is < then hour end
324         // (i.e. 22 to 4)
325         } else if (start > end) {
326             for (int i = start; i<24; i++) {
327                 if (maxIn < counterIn[i]) maxIn = counterIn[i];
328                 if (maxOut < counterOut[i]) maxOut = counterOut[i];
329                 if (minIn > counterIn[i]) minIn = counterIn[i];
330                 if (minOut > counterOut[i]) minOut = counterOut[i];
```

```
331
332                 inTotal += counterIn[i];
333                 outTotal += counterOut[i];
334             }
335             for (int i = 0; i<end; i++) {
336                 if (maxIn < counterIn[i]) maxIn = counterIn[i];
337                 if (maxOut < counterOut[i]) maxOut = counterOut[i];
338                 if (minIn > counterIn[i]) minIn = counterIn[i];
339                 if (minOut > counterOut[i]) minOut = counterOut[i];
340 
341                 inTotal += counterIn[i];
342                 outTotal += counterOut[i];
343             }
344         }
345         else {
346             if (maxIn < counterIn[start]) maxIn = counterIn[start];
347             if (maxOut < counterOut[start]) maxOut = counterOut[start];
348             if (minIn > counterIn[start]) minIn = counterIn[start];
349             if (minOut > counterOut[start]) minOut = counterOut[start];
350 
351             inTotal += counterIn[start];
352             outTotal += counterOut[start];
353         }
354 
355         // save counters data to Stations list
356         auxStations->max_bikesIn    = maxIn;
357         auxStations->max_bikesOut   = maxOut;
358         auxStations->min_bikesIn    = minIn;
359         auxStations->min_bikesOut   = minOut;
360 
361         // calculate average
362         if (start < end) {
363             auxStations->avg_bikesIn    = inTotal/(end-start);
364             auxStations->avg_bikesOut   = outTotal/(end-start);
365         } else if (start == end) {
366             auxStations->avg_bikesIn    = inTotal;
367             auxStations->avg_bikesOut   = outTotal;
368         } if (start > end) {
369             auxStations->avg_bikesIn    = inTotal/((24-start)+end);
370             auxStations->avg_bikesOut    = outTotal/((24-start)+end);
371         }
372         auxStations = auxStations->next;
373     }
374     return stationsList;
375 }
376 
377 
378 // LIST FILTERS
379 
380 /* selectTripsByTime: returns list of trips between hour star
381  *                    and hour end
382  * \param   sourceListHead        the head of the trips list
383  *                                (can be filtered)
384  * \param   hour_start            the start hour for the selectTripsByTime
385  *                                filter
386  * \param   hour_end              the end hour for the
387  *                                selectTripsByTime filter
388  * \return  filteredTripsHead     the head of the trips list, filtered
389  */
390 Trip* selectTripsByTime(Trip * sourceListHead, int hour_start,
391                         int hour_end) {
392     Trip *aux = sourceListHead;
393     Trip *filteredTripsHead = NULL;
394     while (aux != NULL) {
395 
396         // Only save the item to the list if the start hour
```

```
397             // and end hour are within the parameters
398             bool shouldSave = false;
399 
400             // if hour start < hour end, then check if trip is between
401             // time span. if user enters trips from 8 to 9, we take all
402             // trips between 8:00 and 8:59
403             if (hour_start < hour_end) {
404                 if (    (aux->start.hour >= hour_start) &&
405                         (aux->start.hour < hour_end)    &&
406                         (aux->end.hour >= hour_start)   &&
407                         (aux->end.hour < hour_end) )    {
408                     shouldSave = true;
409                 }
410             }
411             // handle scenario: hour start > hour end
412             // if user enters trips from 18 to 17,
413             // we take all trips between 18:00 and 23:59,
414             // and all trips between 0:00 and 17:59
415             else if (hour_start > hour_end) {
416                 if ( ( (aux->start.hour >= hour_start)  &&
417                         (aux->end.hour <= 23) )         ||
418                         ((aux->start.hour >= 0)         &&
419                         (aux->end.hour < hour_end)) )   {
420                     // get all trips between hour_start and 23:59.
421                     // get all trips between 0 and hour_end.
422                     shouldSave = true;
423                 }
424             }
425             // handle scenario: hour start = hour end
426             // if user enters trips from 16 to 16, we take all trips
427             // between 16:00 and 16:59
428             else {
429                 if (    (aux->start.hour == hour_start) &&
430                         (aux->end.hour == hour_end) )   {
431                     shouldSave = true;
432                 }
433             }
434             // save item if it should
435             if (shouldSave) {
436                 filteredTripsHead = copyTripToList(filteredTripsHead, aux);
437             }
438 
439             aux = aux->next;
440         }
441         return filteredTripsHead;
442     }
443 
444     /* selectTripsByDuration: returns list of trips given max duration
445      * \param   sourceListHead        the head of the trips list
446      *                                (can be filtered)
447      * \param   duration              the maximum duration of a trip in
448      *                                seconds
449      * \return  filteredTripsHead     the head of the trips list, filtered
450      */
451     Trip* selectTripsByDuration(Trip * sourceListHead, int duration) {
452         struct Trip *aux = sourceListHead;
453         struct Trip *filteredTripsHead = NULL;
454         while (aux != NULL) {
455             if (aux->duration <= duration) {
456 
457                 filteredTripsHead = copyTripToList(filteredTripsHead, aux);
458             }
459             aux = aux->next;
460         }
461         return filteredTripsHead;
462     }
```

```
463
464
465
466
467  /* selectTripsByDay: returns list of trips given day of week
468   * \param   sourceListHead        the head of the trips list
469   *                                (can be filtered)
470   * \param   selectedDay           an int representing the day of the
471   *                                week (1 monday..7 sunday)
472   * \return  filteredTripsHead     the head of the trips list, filtered
473   */
474  Trip* selectTripsByDay(Trip * sourceListHead, int selectedDay){
475
476      struct Trip *aux = sourceListHead;
477      struct Trip *filteredTripsHead = NULL;
478
479      if (selectedDay == 7) {
480          selectedDay = 0;
481      }
482
483      while (aux != NULL) {
484
485          // Check if day of the current trip is == to the selected day
486          if ((calculateWeekDateFromDate(aux->start.year, aux->start.month,
487              aux->start.day) == selectedDay) ||
488              (calculateWeekDateFromDate(aux->end.year, aux->end.month,
489              aux->end.day) == selectedDay)){
490
491              filteredTripsHead = copyTripToList(filteredTripsHead, aux);
492          }
493          aux = aux->next;
494      }
495      return filteredTripsHead;
496  }
497
498  /* selectTripsByIdStation: returns list of trips given a station ID
499   * \param   sourceListHead        the head of the trips list
500   *                                (can be filtered)
501   * \param   id                    the station ID
502   * \return  filteredTripsHead     the head of the trips list, filtered
503   */
504  Trip* selectTripsByIdStation(Trip * sourceListHead, int id) {
505      int counter=0;
506      struct Trip *aux = sourceListHead;
507      struct Trip *filteredTripsHead = NULL;
508      while (aux != NULL) {
509
510          if ((aux->id_final_station == id)||(aux->id_start_station == id)){
511              filteredTripsHead = copyTripToList(filteredTripsHead, aux);
512              counter++;
513          }
514
515          aux = aux->next;
516      }
517      return filteredTripsHead;
518  }
519
520  // HELPERS
521
522  /* copyTripToList: add element at top of Trip List
523   * \param   filteredTripsHead     the head of the trips list to add
524   *                                the element into
525   * \param   aux                   the trip to be added to the list
526   * \return  filteredTripsHead     the head of the trips list, with
527   *                                the new element at the top
528   */
```

```c
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/
 * \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) {
```

```
595      Station * auxStations = allStations;
596      while (auxStations != NULL) {
597          if (auxStations->id == id) {
598              return auxStations->name;
599          }
600          auxStations = auxStations->next;
601      }
602      return "";
603  }
604
605
```