```c

/* Project created by Sara Marfella IST188316 on May 16, 2017
 * dataManager.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[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);
```

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

```
133                         if (strlen(token)-1 > 8) {
134                             if (strcmp(token, "Registered") !=0) {
135                                 trip->type = REGISTERED;
136                             } else {
137                                 trip->type = CASUAL;
138                             }
139                             // add one to field counter so we skip
140                             // this field
141                             fieldCounter++;
142                             break;
143                         }
144                         strcpy(trip->bike, token);
145                         break;
146                     case 17:
147                         if (strcmp(token, "Registered") == 0) {
148                             trip->type = REGISTERED;
149                         } else {
150                             trip->type = CASUAL;
151                         }
152                         break;
153                     case 18:
154                         trip->year_birthday = atoi(token);
155                         break;
156                     case 19:
157                         if (token[0] == 'M') {
158                             trip->gender = MALE;
159                         } else if (token[0] == 'F') {
160                             trip->gender = FEMALE;
161                         } else {
162                             trip->gender = 0;
163                         }
164                         break;
165                     default: break;
166                 }
167                 // printf ("%s\n",token);
168                 fieldCounter++;
169                 token = strtok (NULL, separators);
170             }
171             // add new trip to linked list
172             trip->next = tripsHead;
173             tripsHead = trip;
174         }
175         fclose(fileOne);
176     }
177     return tripsHead;
178 }
179 
180 // LIST CREATORS
181 
182 /* createRoutesList: returns the list of routes, ordered descendant
183  * \param   tripList              the head of the trips list
184  *                                (can be filtered)
185  * \param   allStations           the head of all stations list
186  * \param   selected_station_id   the ID of the selected station
187  * \return  routes                the head of the routes list
188  */
189 Route * createRoutesList(Trip * tripList, Station * allStations,
190                          int selected_station_id) {
191     Route * routes = NULL;
192     Station * auxStations = allStations;
193 
194     char selected_station_name[ID_SIZE];
195     strcpy(selected_station_name,
196            getStationNameById(selected_station_id, allStations));
197 
198     // foreach station
```

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

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

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

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

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

```
595   */
596  char * getStationNameById(int id, Station * allStations) {
597      Station * auxStations = allStations;
598      while (auxStations != NULL) {
599          if (auxStations->id == id) {
600              return auxStations->name;
601          }
602          auxStations = auxStations->next;
603      }
604      return "";
605  }
606
607
```