```
#include "dataManager.h"
 1
 2
 3
 4
 5
    /* readStationData: parse csv file for stations
     * \param filename path and filename of csv file for stations
 6
                                 list of stations header
7
    * \return stationsHead
 8
9
    Station * readStationData(char *filename){
10
        char line[1024];
        char *token;
11
        char *separators = ",";
12
13
        int lineNumber = 0;
        int fieldCounter = 0;
14
15
        Station * stationsHead = NULL;
16
17
18
        //open file for reading
19
        FILE *fileTwo = fopen( filename, "r" );
        if ( fileTwo == 0 ) {
20
21
            printf( "Error - Could not open stations file: %s\n", filename );
22
            exit(EXIT_FAILURE);
23
24
        else {
25
            while(fgets(line, sizeof line, fileTwo) != NULL){
26
                 / keep line count for convenience
27
                lineNumber++;
                // Split the line into parts
28
29
                token = strtok(line, separators);
30
                // make sure field counter is 0
31
                fieldCounter = 0;
32
                // Allocation of memory
33
                Station* station = (Station*)malloc(sizeof(Station));
                // Skip first line with headers
34
                if (lineNumber != 1) {
35
                    // cycle through fields
36
37
                    while (token != NULL) {
                         // printf("Field: %d\n", fieldCounter);
38
39
                        switch (fieldCounter) {
40
                             case 0: station->id = atoi(token);
41
                                 break;
42
                             case 1: strcpy(station->name, token);
43
                                 break;
44
                             case 2: strcpy(station->full name, token);
45
                                 break;
46
                             case 3: strcpy(station->municipal, token);
47
                                 break;
48
                             case 4: station->latitude = atof(token);
49
                                 break;
50
                             case 5: station->longitude = atof(token);
                                 break;
51
                             case 6:
52
53
                                 if (strcmp(token, "Existing") == 0) {
54
                                     station->status = EXISTING;
55
                                  else {
56
                                     station->status = REMOVED;
57
58
                                 break;
59
                             default: break;
60
                         // printf ("%s\n",token);
61
62
                        fieldCounter++;
63
                        token = strtok (NULL, separators);
64
65
                     // add new trip to linked list
66
                    station->next = stationsHead;
```

```
67
                     stationsHead = station;
 68
 69
70
             fclose(fileTwo);
 71
 72
         return stationsHead;
 73
74
75
     /* readTripsData: parse csv file for trips
     * \param filename path and filename of csv file for trips
76
     * \return tripsHead
77
                               list of trips header
 78
 79
     Trip * readTripsData(char *filename){
 80
         char line[1024];
         char *token;
81
         char *separators = ", /:\n";
82
83
         int lineNumber = 0;
84
         int fieldCounter = 0;
85
86
         Trip * tripsHead = NULL;
 87
88
         //open file for reading
 89
         FILE *fileOne = fopen( filename, "r" );
         if ( fileOne == 0 ) {
90
             printf( "Error - Could not open trips file: %s\n", filename );
91
92
             exit(EXIT_FAILURE);
93
         else {
94
95
96
             while(fgets(line, sizeof line, fileOne) != NULL){
97
                 // keep line count for convenience
98
                 lineNumber++;
99
                 // Split the line into parts
100
                 token = strtok(line, separators);
101
                 // make sure field counter is 0
102
                 fieldCounter = 0;
                 // Allocation of memory
103
                 Trip* trip = (Trip*)malloc(sizeof(Trip));
104
                 // cycle through fields
105
                 while (token != NULL) {
106
107
                     //printf("%d %s\n", fieldCounter, token);
108
                     switch (fieldCounter) {
109
110
                         case 0: trip->id = atoi(token);
                                                                           break;
111
                         case 1:
                                  trip->duration = atoi(token);
                                                                           break;
112
                                  trip->start.month = atoi(token);
                         case 2:
                                                                           break;
113
                         case 3:
                                  trip->start.day = atoi(token);
                                                                           break;
114
                         case 4: trip->start.year = atoi(token);
                                                                           break;
115
                         case 5: trip->start.hour = atoi(token);
                                                                           break;
116
                         case 6:
                                  trip->start.minute = atoi(token);
                                                                           break;
117
                         case 7:
                                  /*"seconds field" to be ignored*/
                                                                           break;
                         case 8:
118
                                  trip->id_start_station = atoi(token);
                                                                          break;
119
                         case 9: trip->end.month = atoi(token);
                                                                           break;
120
                         case 10: trip->end.day = atoi(token);
                                                                           break;
                         case 11: trip->end.year = atoi(token);
121
                                                                           break;
122
                                                                           break;
                         case 12: trip->end.hour = atoi(token);
123
                         case 13: trip->end.minute = atoi(token);
                                                                           break;
                         case 14: /*"seconds field" to be ignored*/
124
                                                                           break;
125
                         case 15: trip->id_final_station = atoi(token); break;
126
                         case 16:
127
                                handle missing bike id on line 609
128
                              if (strlen(token)-1 > 8) {
                                  if (strcmp(token, "Registered") !=0) {
129
130
                                      trip->type = REGISTERED;
131
                                  } else {
132
                                      trip->type = CASUAL;
```

```
133
                                  // add one to field counter so we skip
134
135
                                  // this field
136
                                  fieldCounter++;
137
                                 break;
138
139
                              strcpy(trip->bike, token);
140
                             break;
141
                         case 17:
142
                              if (strcmp(token, "Registered") == 0) {
143
                                  trip->type = REGISTERED;
144
                               else {
145
                                  trip->type = CASUAL;
146
147
                             break;
148
                         case 18:
149
                              trip->year birthday = atoi(token);
150
                             break;
151
                         case 19:
152
                              if (token[0] == 'M') {
153
                                  trip->gender = MALE;
154
                                else if (token[0] == 'F') {
155
                                  trip->gender = FEMALE;
156
                                else {
157
                                  trip->gender = 0;
158
159
                             break;
160
                         default: break;
161
162
                      // printf ("%s\n",token);
163
                     fieldCounter++;
164
                     token = strtok (NULL, separators);
165
166
                 // add new trip to linked list
167
                 trip->next = tripsHead;
168
                 tripsHead = trip;
169
170
             fclose(fileOne);
171
172
         return tripsHead;
173
174
175
176
177
     /* createRoutesList: returns the list of routes, ordered descendant
     * \param tripList
178
                                        the head of the trips list
179
                                        (can be filtered)
180
     * \param allStations
                                        the head of all stations list
181
      * \param selected_station_id the ID of the selected station
      * \return routes
                                        the head of the routes list
182
183
184 Route * createRoutesList(Trip * tripList, Station * allStations,
185
                               int selected_station_id) {
186
         Route * routes = NULL;
187
         Station * auxStations = allStations;
188
189
         char selected_station_name[7];
190
         strcpy(selected_station_name,
191
                getStationNameById(selected_station_id, allStations));
192
193
         // foreach station
194
         while (auxStations != NULL) {
195
196
             // initialize route counters
197
             int tripsIn = 0;
198
             int tripsOut = 0;
```

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

264

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

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

```
396
             // if hour start < hour end, then check if trip is between
397
             // time span. if user enters trips from 8 to 9, we take all
                trips between 8:00 and 8:59
398
399
             if (hour_start < hour_end) {</pre>
400
                         (aux->start.hour >= hour_start) &&
                 if (
401
                         (aux->start.hour < hour_end)</pre>
402
                         (aux->end.hour >= hour_start)
                                                          &&
403
                         (aux->end.hour < hour_end) )</pre>
404
                     shouldSave = true;
405
406
             // handle scenario: hour start > hour end
407
408
             // if user enters trips from 18 to 17,
409
             // we take all trips between 18:00 and 23:59,
410
             // and all trips between 0:00 and 17:59
411
             else if (hour_start > hour_end) {
412
                 if ( (aux->start.hour >= hour start) &&
                         (aux->end.hour <= 23) )
413
                                                          414
                         ((aux->start.hour >= 0)
                                                          &&
415
                         (aux->end.hour < hour_end)) )</pre>
416
                     // get all trips between hour_start and 23:59.
417
                     // get all trips between 0 and hour_end.
418
                     shouldSave = true;
419
420
421
             // handle scenario: hour start = hour end
             // if user enters trips from 16 to 16, we take all trips
422
             // between 16:00 and 16:59
423
             else {
424
425
                 if (
                         (aux->start.hour == hour_start) &&
426
                         (aux->end.hour == hour_end) )
427
                     shouldSave = true;
428
429
             // save item if it should
430
431
             if (shouldSave) {
432
                 filteredTripsHead = copyTripToList(filteredTripsHead, aux);
433
434
435
             aux = aux->next;
436
437
        return filteredTripsHead;
438
439
440
     /* selectTripsByDuration: returns list of trips given max duration
441
                                       the head of the trips list
       \param sourceListHead
442
                                        (can be filtered)
443
     * \param duration
                                        the maximum duration of a trip in
444
                                        seconds
                                      the head of the trips list, filtered
445
      * \return filteredTripsHead
446
     Trip* selectTripsByDuration(Trip * sourceListHead, int duration) {
447
448
         struct Trip *aux = sourceListHead;
449
         struct Trip *filteredTripsHead = NULL;
450
         while (aux != NULL) {
             if (aux->duration <= duration) {</pre>
451
452
453
                 filteredTripsHead = copyTripToList(filteredTripsHead, aux);
454
455
             aux = aux->next;
456
457
         return filteredTripsHead;
458
459
460 /* selectTripsByDay: returns list of trips given day of week
461
     * \param sourceListHead the head of the trips list
```

```
462
                                        (can be filtered)
      * \param
463
                                        an int representing the day of the
               selectedDay
464
                                        week (1 monday..7 sunday)
465
      * \return filteredTripsHead
                                       the head of the trips list, filtered
466
467
    Trip* selectTripsByDay(Trip * sourceListHead, int selectedDay){
468
469
         struct Trip *aux = sourceListHead;
470
         struct Trip *filteredTripsHead = NULL;
471
472
         if (selectedDay == 7) {
473
             selectedDay = 0;
474
475
476
         while (aux != NULL) {
477
478
             // Check if day of the current trip is == to the selected day
479
             if ((calculateWeekDateFromDate(aux->start.year, aux->start.month,
480
                 aux->start.day) == selectedDay)
481
                 (calculateWeekDateFromDate(aux->end.year, aux->end.month,
482
                 aux->end.day) == selectedDay)){
483
484
                 filteredTripsHead = copyTripToList(filteredTripsHead, aux);
485
486
             aux = aux->next;
487
488
         return filteredTripsHead;
489
490
491
     /* selectTripsByIdStation: returns list of trips given a station ID
                                        the head of the trips list
492
     * \param sourceListHead
493
                                        (can be filtered)
      * \param
494
                                        the station ID
                 id
      * \return filteredTripsHead
495
                                       the head of the trips list, filtered
496
      * /
497
     Trip* selectTripsByIdStation(Trip * sourceListHead, int id) {
498
         int counter=0;
499
         struct Trip *aux = sourceListHead;
500
         struct Trip *filteredTripsHead = NULL;
501
         while (aux != NULL) {
502
             if ((aux->id final station == id) | (aux->id start station == id)){
503
504
                 filteredTripsHead = copyTripToList(filteredTripsHead, aux);
505
                 counter++;
506
507
508
             aux = aux->next;
509
510
         return filteredTripsHead;
511
512
513
514
515
     /* copyTripToList: add element at top of Trip List
516
                                        the head of the trips list to add
       \param filteredTripsHead
517
                                        the element into
      * \param aux
518
                                        the trip to be added to the list
519
      * \return filteredTripsHead
                                        the head of the trips list, with
520
                                        the new element at the top
521
522
     Trip* copyTripToList(Trip * filteredTripsHead, Trip * aux) {
523
524
         Trip* trip = (Trip*)malloc(sizeof(Trip));
525
526
         trip->id = aux->id;
527
         strcpy(trip->bike, aux->bike);
```

```
528
         trip->duration = aux->duration;
529
         trip->end = aux->end;
530
         trip->start = aux->start;
531
         trip->gender = aux->gender;
532
         trip->id_final_station = aux->id_final_station;
533
         trip->id_start_station = aux->id_start_station;
534
         trip->type = aux->type;
535
         trip->year_birthday = aux->year_birthday;
536
537
         trip->next = filteredTripsHead;
538
         filteredTripsHead = trip;
539
540
         return filteredTripsHead;
541
    }
542
543
    /* calculateWeekDateFromDate: Calculate the day of the week this
544
                                   current trip was in
     * -Source:
545
546
stackoverflow.com/questions/6054016/c-program-to-find-day-of-week-qiven-date
                         year (4 digits)
547
      * \param y
     * \param m
548
                         month (2 digits)
     * \param d
549
                         day (2 digits)
     * \return weekday number 1 to 7 with 1 = monday and 7 = sunday
550
551
552
     int calculateWeekDateFromDate(int y, int m, int d) {
553
       int weekday=(d+=m<3 ? y-- : y - 2, 23*m/9 +d+ 4 + y/4- y/100 + y/400)%7;
554
       return weekday;
555
556
557
     /* sortedInsert: insert a route in the correct order
                               the head of the list to add the route into
558
     * \param head_ref
      * \param new_node
                                the new route to add to the list
559
560
     * /
561
    void sortedInsert(Route** head_ref, Route* new_node) {
562
         Route* current;
563
         // handle scenario for the head end
         if (*head_ref == NULL | (*head_ref)->total <= new_node->total) {
564
             new_node->next = *head_ref;
565
566
             *head_ref = new_node;
567
568
         else
569
570
             // find the node before the point of insert
571
             current = *head ref;
572
             while (current->next != NULL &&
573
                    current->next->total > new_node->total) {
574
                 current = current->next;
575
576
             new_node->next = current->next;
577
             current->next = new_node;
         }
578
579
580
581
     /* getStationNameById: Get a Station Name by its ID
582
                                     the ID of the station to look for
        \param id
583
      * \param allStations
                                     the head of all stations list
584
      * \return auxStations->name
                                   containing the station name,
585
                                     or an empty string
586
                                     if no station was found
      * /
587
588
    char * getStationNameById(int id, Station * allStations) {
589
         Station * auxStations = allStations;
590
         while (auxStations != NULL)
591
             if (auxStations->id == id) {
592
                 return auxStations->name;
```

```
593
594
595
596
597
598
599

}

auxStations = auxStations->next;

preturn "";
```