```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;
```

```c
 67                          stationsHead = station;
 68                  }
 69              }
 70              fclose(fileTwo);
 71          }
 72      return stationsHead;
 73  }
 74
 75  /* readTripsData: parse csv file for trips
 76   * \param    filename        path and filename of csv file for trips
 77   * \return   tripsHead     list of trips header
 78   */
 79  Trip * readTripsData(char *filename){
 80      char line[1024];
 81      char *token;
 82      char *separators = ", /:\n";
 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" );
 90      if ( fileOne == 0 ) {
 91          printf( "Error - Could not open trips file: %s\n", filename );
 92          exit(EXIT_FAILURE);
 93      }
 94      else {
 95          // read each line
 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;
103              // Allocation of memory
104              Trip* trip = (Trip*)malloc(sizeof(Trip));
105              // cycle through fields
106              while (token != NULL) {
107
108                  //printf("%d %s\n", fieldCounter, token);
109                  switch (fieldCounter) {
110                      case 0:  trip->id = atoi(token);              break;
111                      case 1:  trip->duration = atoi(token);        break;
112                      case 2:  trip->start.month = atoi(token);     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;
118                      case 8:  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;
121                      case 11: trip->end.year = atoi(token);        break;
122                      case 12: trip->end.hour = atoi(token);        break;
123                      case 13: trip->end.minute = atoi(token);      break;
124                      case 14: /*"seconds field" to be ignored*/    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) {
129                              if (strcmp(token, "Registered") !=0) {
130                                  trip->type = REGISTERED;
131                              } else {
132                                  trip->type = CASUAL;
```

```c
133                            }
134                            // add one to field counter so we skip
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 // LIST CREATORS
176
177 /* createRoutesList: returns the list of routes, ordered descendant
178  * \param    tripList              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
182  * \return   routes               the head of the routes list
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;
```

```c
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                    }
210                    // count trips from selected station to current station
211                    else if (auxTrips->id_final_station == auxStations->id) {
212                        tripsIn++;
213                    }
214                    // make sure we only update one counter if the trip is
215                    // from and to the same station
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;
229                strcpy(routeIn->name_final_station, selected_station_name);
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;
242                strcpy(routeOut->name_start_station, selected_station_name);
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
256  * \param   filtered_hour_end   the end hour for the selectTripsByTime
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
```

```c
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;
297            if (filter_hour_start != -1 && filter_hour_end != -1) {
298
299                start = filter_hour_start;
300                end = filter_hour_end;
301
302                maxIn = counterIn[filter_hour_start];
303                minIn = counterIn[filter_hour_start];
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) {
310                for (int i = start; i<end; i++){
311                    if (maxIn < counterIn[i]) maxIn = counterIn[i];
312                    if (maxOut < counterOut[i]) maxOut = counterOut[i];
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];
325                    if (maxOut < counterOut[i]) maxOut = counterOut[i];
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++) {
333                      if (maxIn < counterIn[i]) maxIn = counterIn[i];
334                      if (maxOut < counterOut[i]) maxOut = counterOut[i];
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];
344                  if (maxOut < counterOut[start]) maxOut = counterOut[start];
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;
355          auxStations->min_bikesIn    = minIn;
356          auxStations->min_bikesOut   = minOut;
357
358          // calculate average
359          if (start < end) {
360              auxStations->avg_bikesIn    = inTotal/(end-start);
361              auxStations->avg_bikesOut   = outTotal/(end-start);
362          } else if (start == end) {
363              auxStations->avg_bikesIn    = inTotal;
364              auxStations->avg_bikesOut   = outTotal;
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  // LIST FILTERS
376
377  /* selectTripsByTime: returns list of trips between hour star
378   *                    and hour end
379   * \param   sourceListHead      the head of the trips list
380   *                              (can be filtered)
381   * \param   hour_start          the start hour for the selectTripsByTime
382   *                              filter
383   * \param   hour_end            the end hour for the
384   *                              selectTripsByTime filter
385   * \return  filteredTripsHead   the head of the trips list, filtered
386   */
387  Trip* selectTripsByTime(Trip * sourceListHead, int hour_start,
388                          int hour_end) {
389      Trip *aux = sourceListHead;
390      Trip *filteredTripsHead = NULL;
391      while (aux != NULL) {
392
393          // Only save the item to the list if the start hour
394          // and end hour are within the parameters
395          bool shouldSave = false;
396
```

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

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

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

```
595                return auxStations->name;
596            }
597        auxStations = auxStations->next;
598    }
599    return "";
600 }
601
602
```