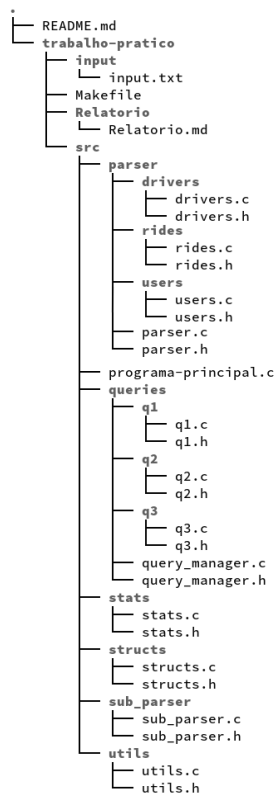


- **structs** - conjunto de definições de estruturas de dados e respetivos métodos para acesso às estruturas anónimas dos users, drivers e rides;
- **parser** - parsing de ficheiros .csv (de dados);
- **sub parser** - parsing do ficheiro .txt (de comandos) e encaminhamento para o processador de queries;
- **queries** - processamento dos dados de acordo com o input, e criação do output;
- **stats** - conjus dos stats dos users, drivers e rides;
- **utils** - conjunto de funções de utilidade geral.



## Programa principal

programa-principal.c

```
#include "../sub_parser/sub_parser.h"
#include "../structs/structs.h"
#include "../parser/parser.h"
#include "../stats/stats.h"

int main (int argc, char **argv) {

    if (argc >= 2) {
        HASH *hash = create_hash();
        STATS *stats = create_stats();

        parser(argv[1], hash);
        calc_stats(hash, stats);
        sub_parser(stats, hash, argv[2]);

        destroy_hash(hash);
        destroy_stats(stats);
    }
    return 0;
}
```

Nesta parte do programa são chamadas as funções centrais do programa tais como o parser, as funções para calcular as estatísticas, para ler e executar os comandos e as respetivas funções para libertação de memória.

## Estruturas opacas

- **HASH:** esta é uma estrutura opaca que contém os pointers para as hash tables em que são guardadas todas as informações relativas aos drivers, users e rides.
- **DRIVER, USER e RIDE:** são estruturas opacas que contém a informação relativa a somente um driver/user/rides, respetivamente.
- **STATS:** esta é uma estrutura opaca que contém os pointers para as hash tables em que são guardadas todas as estatísticas relativas aos drivers, users e rides.

## Structs

structs.h

```
typedef struct ht HASH;

typedef struct usr USER;
typedef struct drv DRIVER;
typedef struct rd RIDE;

HASH *create_hash ();
void destroy_hash ();

void *get_drivers_hash (HASH *hash);
void *get_users_hash (HASH *hash);
void *get_rides_hash (HASH *hash);

USER *create_user
(char *username,
 char *name,
 char *gender,
 char *birth_date,
 char *account_creation,
 char *pay_method,
 char *account_status
);

char *get_user_username (const USER *user);
char *get_user_name (const USER *user);
char *get_user_gender (const USER *user);
char *get_user_birth_date (const USER *user);
char *get_user_account_creation (const USER *user);
char *get_user_pay_method (const USER *user);
char *get_user_account_status (const USER *user);

void set_user_username (USER *user, char *username );
void set_user_name (USER *user, char *name );
void set_user_gender (USER *user, char *gender );
void set_user_birth_date (USER *user, char *birth_date );
void set_user_account_creation (USER *user, char *account_creation);
void set_user_pay_method (USER *user, char *pay_method );
void set_user_account_status (USER *user, char *account_status );
```

Para manter o encapsulamento, cada vez que se é um campo de um driver, user ou ride é passada uma cópia, seja ela int, double (no caso da distância e gorjeta, respectivamente), ou uma cópia da string.

No caso dos getters para as hash são simplesmente devolvidos os pointers, pois são estritamente necessários para o acesso às hash tables.

structs.c

```
typedef struct rd {
    char *id;
    char *date;
    char *driver;
    char *user;
    char *city;
    char *comment;
    int distance;
    int score_user;
    int score_driver;
    double tip;
}
```

```

} RIDE;

typedef struct ht {
    void *users; // key: char *username || value: USER*
    void *drivers; // key: char *driver_id || value: DRIVER*
    void *rides; // key: char *ride_id || value: RIDE*
} HASH;

RIDE *create_ride
(char *id,
 char *date,
 char *driver,
 char *user,
 char *city,
 char *distance,
 char *score_user,
 char *score_driver,
 char *tip,
 char *comment
)
{
    RIDE *ride = malloc(sizeof(RIDE));

    ride->id          = strdup(id);
    ride->date         = strdup(date);
    ride->driver       = strdup(driver);
    ride->user         = strdup(user);
    ride->city         = strdup(city);
    ride->distance     = atoi(distance);
    ride->score_user   = atol(score_user);
    ride->score_driver = atol(score_driver);
    ride->tip          = atof(tip);
    ride->comment      = strdup(comment);

    return ride;
}

void destroy_ride (void *data) {
    RIDE *ride = data;

    free(ride->id);
    free(ride->date);
    free(ride->driver);
    free(ride->user);
    free(ride->city);
    free(ride->comment);

    free(ride);
}

HASH *create_hash () {
    HASH *hash = malloc(sizeof(HASH));

    hash->users  = g_hash_table_new_full(g_str_hash, g_str_equal, free, destroy_user );
    hash->drivers = g_hash_table_new_full(g_str_hash, g_str_equal, free, destroy_driver);
    hash->rides  = g_hash_table_new_full(g_str_hash, g_str_equal, free, destroy_ride );

    return hash;
}

void destroy_hash (HASH *hash) {
    g_hash_table_destroy(hash->users );

```

```

    g_hash_table_destroy(hash->drivers);
    g_hash_table_destroy(hash->rides );

    free(hash);
}

char *get_ride_id      (const RIDE *ride) {return strdup(ride->id )    ;}
char *get_ride_date    (const RIDE *ride) {return strdup(ride->date )  ;}
char *get_ride_driver  (const RIDE *ride) {return strdup(ride->driver );}
char *get_ride_user    (const RIDE *ride) {return strdup(ride->user )  ;}
char *get_ride_city    (const RIDE *ride) {return strdup(ride->city )   ;}
int  get_ride_distance (const RIDE *ride) {return ride->distance      ;}
int  get_ride_score_user (const RIDE *ride) {return ride->score_user   ;}
int  get_ride_score_driver (const RIDE *ride) {return ride->score_driver ;}
double get_ride_tip    (const RIDE *ride) {return ride->tip           ;}
char *get_ride_comment (const RIDE *ride) {return strdup(ride->comment );}

// Neste caso os setters são um bocado inúteis, pois todas as suas informações não deverão mudar durante
a execução do programa
void set_ride_id      (RIDE *ride, char *id)      {free(ride->id);      ride->id      =
id;          }
void set_ride_date    (RIDE *ride, char *date)    {free(ride->date);    ride->date    =
date;        }
void set_ride_driver  (RIDE *ride, char *driver)  {free(ride->driver);  ride->driver  =
driver;      }
void set_ride_user    (RIDE *ride, char *user)    {free(ride->user);    ride->user    =
user;        }
void set_ride_city    (RIDE *ride, char *city)    {free(ride->city);    ride->city    =
city;        }
void set_ride_distance (RIDE *ride, int  distance) {                ride->distance =
distance;    }
void set_ride_score_user (RIDE *ride, int  score_user) {                ride->score_user =
score_user;  }
void set_ride_score_driver (RIDE *ride, int  score_driver) {                ride->score_driver =
score_driver; }
void set_ride_tip      (RIDE *ride, double tip)    {                ride->tip      =
tip;        }
void set_ride_comment  (RIDE *ride, char *comment) {free(ride->comment); ride->comment =
comment;    }

```

Como podemos ver todas as funções aqui implementadas são bastante triviais.

## Parser

```
void parser (char *path, HASH *hash);
```

parser.c

```

void parser (char *path, HASH *hash) {
    char *path_users = concat(path, "/users.csv");
    parser_users(path_users, hash);
    free(path_users);

    char *path_drivers = concat(path, "/drivers.csv");
    parser_drivers(path_drivers, hash);
    free(path_drivers);

    char *path_rides = concat(path, "/rides.csv");
    parser_rides(path_rides, hash);
}

```

```
    free(path_rides);  
}
```

Aqui simplesmente são tratados os paths e chamadas os respetivos parsers para os files.

Só irei demonstrar um dos seguintes parsers para os files, pois as diferenças entre ambos são negligíveis.

### Parser Drivers

drivers.c

```
void parser_drivers (char *path, HASH *hash) {  
    FILE *fp = fopen(path, "r");  
    if (!fp) assert(0);  
  
    void *drivers_hash = get_drivers_hash(hash);  
  
    char *line;  
    size_t len = 0;  
    getline(&line, &len, fp); // remover a primeira linha  
    while (getline(&line, &len, fp) != -1) {  
        char *aux = line;  
        char *id = strtok(aux, ";\\n");  
        char *name = strtok(aux, ";\\n");  
        char *birth_date = strtok(aux, ";\\n");  
        char *gender = strtok(aux, ";\\n");  
        char *car_class = strtok(aux, ";\\n");  
        char *license_plate = strtok(aux, ";\\n");  
        char *city = strtok(aux, ";\\n");  
        char *account_creation = strtok(aux, ";\\n");  
        char *account_status = strtok(aux, ";\\n");  
        DRIVER *driver = create_driver(id,  
                                       name,  
                                       birth_date,  
                                       gender,  
                                       car_class,  
                                       license_plate,  
                                       city,  
                                       account_creation,  
                                       account_status);  
        g_hash_table_insert(drivers_hash, get_driver_id(driver), driver);  
    }  
  
    free(line);  
    fclose(fp);  
}
```

Como podemos ver simplesmente é-se percorrido o ficheiro `drivers.csv` e é-se usada a função do `structs.h` para criar o `DRIVER` e enviar para a hash table.

### Stats

stats.h

```
typedef struct st STATS;  
  
typedef struct usr_st USER_STATS;  
typedef struct drv_st DRIVER_STATS;  
typedef struct rd_st RIDE_STATS;  
  
STATS *create_stats () ;
```

```

void destroy_stats (STATS *stats);

void calc_stats (HASH *hash, STATS *stats);

void *get_drivers_stats_hash(STATS *stats);
void *get_users_stats_hash (STATS *stats);

char *get_user_stats_username      (const USER_STATS *user);
double get_user_stats_avaliacao_media (const USER_STATS *user);
double get_user_stats_total_gasto    (const USER_STATS *user);
int    get_user_stats_numero_viagens (const USER_STATS *user);
int    get_user_stats_distancia_viajada (const USER_STATS *user);
char *get_user_stats_ultima_viagem  (const USER_STATS *user);
char *get_user_stats_account_status (const USER_STATS *user);

```

Funções bastante simples e triviais, vale mais a pena olhar para o `stats.c` a seguir com as implementações das funções.

```

typedef struct usr_st {
    char *username;

    double avaliacao_media;
    double total_gasto;
    int    numero_viagens;
    int    distancia_viajada;
    char *ultima_viagem;
    char *account_status;
} USER_STATS;

typedef struct rd_st {
    char *id;
} RIDE_STATS;

typedef struct st {
    void *user_stats;
    void *driver_stats;
    void *ride_stats;
} STATS;

typedef struct {
    HASH *hash;
    STATS *stats;
} AUX_STRUCT;

STATS *create_stats () {
    STATS *stats = malloc(sizeof(STATS));

    stats->user_stats = g_hash_table_new_full(g_str_hash, g_str_equal, free, destroy_user_stats );
    stats->driver_stats = g_hash_table_new_full(g_str_hash, g_str_equal, free,
destroy_driver_stats);
    stats->ride_stats = g_hash_table_new_full(g_str_hash, g_str_equal, free, destroy_driver_stats);

    return stats;
}

void destroy_stats (STATS *stats) {
    g_hash_table_destroy(stats->user_stats);
    g_hash_table_destroy(stats->driver_stats);
    g_hash_table_destroy(stats->ride_stats);
}

```

```

USER_STATS *create_user_stats (char *username, char *account_status) {
    USER_STATS *user_stats = malloc(sizeof(USER_STATS));

    user_stats->username      = strdup(username);
    user_stats->avaliacao_media = 0;
    user_stats->total_gasto    = 0;
    user_stats->numero_viagens = 0;
    user_stats->distancia_viajada = 0;
    user_stats->ultima_viagem   = strdup("00/00/0000");
    user_stats->account_status  = strdup(account_status);
    return user_stats;
}

void destroy_user_stats (void *a) {
    USER_STATS *user_stats = a;

    free(user_stats->ultima_viagem);
    free(user_stats->username);
    free(user_stats->account_status);

    free(user_stats);
}

void update_driver_stats
(DRIVER_STATS *driver_stats,
 int score_driver,
 int distance,
 char *car_class,
 char *ultima_viajem
)
{
    driver_stats->numero_viagens = driver_stats->numero_viagens + 1;
    driver_stats->avaliacao_media = (driver_stats->avaliacao_media * (double) (driver_stats->numero_viagens - 1) + score_driver) / (double) driver_stats->numero_viagens;
    driver_stats->distancia_viajada = driver_stats->distancia_viajada + distance;

    if (date_comp(driver_stats->ultima_viagem, ultima_viajem) < 0)
        strncpy(driver_stats->ultima_viagem, ultima_viajem, 11);

    if (*car_class == 'b')
        driver_stats->total_auferido = driver_stats->total_auferido + 3.25 + (0.62 * (double) distance);
    else if (*car_class == 'g')
        driver_stats->total_auferido = driver_stats->total_auferido + 4.00 + (0.79 * (double) distance);
    else if (*car_class == 'p')
        driver_stats->total_auferido = driver_stats->total_auferido + 5.20 + (0.94 * (double) distance);
}

void ride_iter (void *key, void *value, void *user_data) {
    (void) key;
    // ...
    DRIVER *driver = g_hash_table_lookup(get_drivers_hash(hash), driver_id);
    char *car_class = get_driver_car_class(driver);

    DRIVER_STATS *driver_stats = g_hash_table_lookup(stats->driver_stats, driver_id);
    if (!driver_stats) {
        char *account_status = get_driver_account_status(driver);
        driver_stats = create_driver_stats(driver_id, account_status);
        free(account_status);
    }
}

```



```

        g_hash_table_insert(stats->driver_stats, get_ride_driver(ride), driver_stats);
    }
    update_driver_stats(driver_stats,
                        score_driver,
                        distance,
                        car_class,
                        date);

    USER_STATS *user_stats = g_hash_table_lookup(stats->user_stats, user_username);
    if (!user_stats) {
        USER *user = g_hash_table_lookup(get_users_hash(hash), user_username);
        char *account_status = get_user_account_status(user);
        user_stats = create_user_stats(user_username, account_status);
        free(account_status);
        g_hash_table_insert(stats->user_stats, get_ride_user(ride), user_stats);
    }
    update_user_stats(user_stats,
                    score_user,
                    distance,
                    car_class,
                    date);

    free(driver_id);
    free(user_username);
    free(date);
    free(car_class);
}

void calc_stats (HASH *hash, STATS *stats) {
    AUX_STRUCT aux;
    aux.hash = hash;
    aux.stats = stats;
    g_hash_table_foreach(get_rides_hash(hash), ride_iter, &aux);
}

// Getters para acessar aos campos das estatísticas
char *get_user_stats_username (const USER_STATS *user) {return strdup(user->username) ;}
char *get_user_stats_ultima_viagem (const USER_STATS *user) {return strdup(user->ultima_viagem) ;}
char *get_user_stats_account_status (const USER_STATS *user) {return strdup(user->account_status);}
double get_user_stats_avaliacao_media (const USER_STATS *user) {return user->avaliacao_media ;}
double get_user_stats_total_gasto (const USER_STATS *user) {return user->total_gasto ;}
int get_user_stats_numero_viagens (const USER_STATS *user) {return user->numero_viagens ;}
int get_user_stats_distancia_viajada (const USER_STATS *user) {return user->distancia_viajada ;}

void *get_drivers_stats_hash(STATS *stats) {return stats->driver_stats;}
void *get_users_stats_hash (STATS *stats) {return stats->user_stats ;}

```

Nesta parte do código são calculadas as estatísticas para os USER\_STATS e DRIVER\_STATS, sendo que as RIDE\_STATS não contém informação individual relevante.

Para tal efeito é-se iterada sobre a hash table das rides com a `iter_rides` e com essa informação chamamos as `update_user_stats` e `update_driver_stats`.

## Sub Parser

sub\_parser.c

```

void sub_parser (STATS *stats, HASH *hash, char *path) {
    char *filepath = concat(path, "input.txt");
    FILE *fp = fopen(filepath, "r");
    free(filepath);

```

```

if (!fp) assert(0);

char *line;
size_t len = 0;
for (int i = 1; getline(&line, &len, fp) != -1; i++) {
    line = strtok(&line, "\n");
    char *str = line;
    char *num = strtok(&str, " ");
    int q = atoi(num);

    query_manager(stats, hash, path, q, i, str);
}

free(line);
fclose(fp);
}

```

Esta é a única função do `sub_parser.c`, que lê o ficheiro dos comandos linha a linha e que chama o `query_manager`, que por sua vez interpreta os comandos.

## Query Manager

| `query_manager.c`

```

#include "../q1/q1.h"
#include "../q2/q2.h"
#include "../q3/q3.h"

void query_manager (STATS *stats, HASH *hash, char *path, int query, int i, char *line) {
    (void) path;
    char *filepath = "../Resultados/";

    switch(query) {
        case 1:
            query_1 (stats, hash, filepath, i, line);
            break;
        case 2:
            query_2 (stats, hash, filepath, i, line);
            break;
        case 3:
            query_3 (stats, hash, filepath, i, line);
            break;
        default:
            break;
    }
}

```

Esta função simplesmente chama a query correspondente ao comando com a informação devida.

## Query 1

```

void query_1 (STATS *stats, HASH *hash, char *path, int i, char *str) {
    char *file = malloc(sizeof(char) * 25);
    sprintf(file, "command%d_output.txt", i);
    char *filepath = concat(path, file);
    free(file);

    FILE *fp = fopen(filepath, "w+");
    free(filepath);

    if ((*str >= '0') && (*str <= '9')) { // driver

```

```

DRIVER *driver = g_hash_table_lookup(get_drivers_hash(hash), str);
DRIVER_STATS *driver_stats = g_hash_table_lookup(get_drivers_stats_hash(stats), str);

char *nome          = get_driver_name(driver);
char  genero         = get_driver_gender(driver);
char *idade_str      = get_driver_birth_date(driver);
int   idade          = date_age(idade_str);

double avaliacao_media = get_driver_stats_avaliacao_media(driver_stats);
int    numero_viagens  = get_driver_stats_numero_viagens(driver_stats);
double total_auferido  = get_driver_stats_total_auferido(driver_stats);

fprintf(fp, "%s;%c;%d;%.3f;%d;%.3f", nome, genero, idade, avaliacao_media, numero_viagens,
total_auferido);

free(nome);
free(idade_str);
} else { // user
USER *user = g_hash_table_lookup(get_users_hash(hash), str);
USER_STATS *user_stats = g_hash_table_lookup(get_users_stats_hash(stats), str);

char *nome          = get_user_name(user);
char  genero         = get_user_gender(user);
char *idade_str      = get_user_birth_date(user);
int   idade          = date_age(idade_str);

double avaliacao_media = get_user_stats_avaliacao_media(user_stats);
int    numero_viagens  = get_user_stats_numero_viagens(user_stats);
double total_gasto     = get_user_stats_total_gasto(user_stats);

fprintf(fp, "%s;%c;%d;%.3f;%d;%.3f", nome, genero, idade, avaliacao_media, numero_viagens,
total_gasto);

free(nome);
free(idade_str);
}
fclose(fp);
}

```

Esta função cria o ficheiro, interpreta os argumentos dados, neste caso o id do DRIVER ou o username do USER e devolve a informação necessária.

Para isto, acessa tanto as estatísticas como as informações fornecidas pelas hash tables preenchidas pelos parsers.

## Query 2 & 3

Como ambas as queries são muito parecidas, decidimos juntar as duas num único ponto.

Falaremos apenas da Query 2.

```

// ...

void query_2 (STATS *stats, HASH *hash, char *path, int i, char *line) {
char *file = malloc(sizeof(char) * 25);
sprintf(file, "command%d_output.txt", i);
char *filepath = concat(path, file);
free(file);

FILE *fp = fopen(filepath, "w+");
free(filepath);

int N = atoi(line);

```

```

GList *list = g_hash_table_get_values(get_drivers_stats_hash(stats));
list = g_list_sort(list, q2_comp_func_drivers);
list = list_remove_all_inactive_drivers(list);

GList *aux = list;
for (int j = 0; (j < N) && aux; j++) {
    DRIVER_STATS *driver_stats = g_list_nth_data(aux, 0);

    char *id = get_driver_stats_id(driver_stats);
    double avaliacao_media = get_driver_stats_avaliacao_media(driver_stats);

    DRIVER *driver = g_hash_table_lookup(get_drivers_hash(hash), id);
    char *nome = get_driver_name(driver);

    fprintf(fp, "%s;%s;%.3f\n", id, nome, avaliacao_media);

    free(id);
    free(nome);

    // g_list_next(aux);
    aux = aux->next;
}

fclose(fp);
g_list_free(list);
}

```

Em ambas as queries, 2 e 3, usamos a função `g_hash_table_get_values` para nos devolver uma linked list com todos os values da hash table dos drivers/users, respetivamente.

Com isto podemos usar a função `g_list_sort` definida na Glib para rapidamente ordenar a linked list pelos critérios que nos interessam com o auxílio da função `q2_comp_func_drivers` / `q3_comp_func_users`.

Depois removemos as contas inválidas da linked list e imprimimos para o ficheiro o número de users ou drivers pedidos na query.

## Observações finais

Nesta primeira fase do trabalho foi explorada a utilização do encapsulamento de estruturas, de modularidade de ficheiros e do planeamento da estrutura do código. Desta maneira conseguimos criar um código mais versátil, robusto e mais facilmente expansível, caso seja necessário implementar novas funcionalidades(, tal como será necessário na próxima fase).

Acredito que conseguimos escrever um código sensível e competente, com falhas de memória mínimas e com um tempo de execução sempre inferior a 2 segundos, em todos os dispositivos que este código foi testado.

#LI3

#C

#SoftwareEngineering

#uni