IPCA

Escola superior de Tecnologia

Projeto de Programação Imperativa e Laboratórios de Informática

Autores:
Diogo Bernardo ***21144***
Diogo Borges ***23083***
João Ribeiro ***23795***
João Gabriel ***18355***

Docentes:
João Carlos Silva
Patrícia Leite

submissão do projeto para
o curso de Engenharia de Sistemas Informáticos

ESI

Escola Superior de Tecnologia
github

January 23, 2022

IPCA

Resumo

github

Engenharia de Sistemas Informáticos

Projeto de Programação Imperativa e Laboratórios de Informática

por Diogo Bernardo **21144**
Diogo Borges **23083**
João Ribeiro **23795**
João Gabriel **18355**

Trabalho em cargo da disciplina de Programação Imperativa e Laboratórios de Informática.

Objetivos do projeto

Apresentamos em síntese os nossos objetivos na realização do projeto:

- Apresentar um programa de fácil utilização.
- Apresentar uma solução para a gestão de meios de mobilidade elétrica num contexto de uma smart-city, considerando que os meios de mobilidade elétrica serão sempre levantados e entregues pelos utilizadores no mesmo local.
- Agilizar e facilitar a gestão de ganhos.
- Sedimentar o nosso conhecimento a nível das linguagens aplicadas.
- Projetar o nosso trabalho em grupo.
- Aprender a utilizar ferramentas que facilitam o trabalho em grupo.

Contents

Resumo iii

Objetivos do projeto v

1 Algoritmo 1
- 1.1 Algoritmo . . . 1
- 1.2 Opção de algoritmo "Vetor" . . . 1
  - 1.2.1 Vantagens de um array dinâmico . . . 1
  - 1.2.2 Desvantagens de um array dinâmico . . . 1
- 1.3 Estruturas de dados . . . 1
  - 1.3.1 Veículos . . . 2
  - 1.3.2 Pedidos de utilização . . . 2
- 1.4 implementação do Vetor . . . 2

2 makefile 6

3 Validação de dados 7
- 3.1 síntese . . . 7
- 3.2 código . . . 7
- 3.3 interpretação . . . 9

4 Leitura de ficheiros 10
- 4.1 funções . . . 10
- 4.2 codigo . . . 10

5 Utilitarios 13
- 5.1 Logger . . . 13
  - 5.1.1 header . . . 13
- 5.2 Teste . . . 15
  - 5.2.1 header . . . 15
  - 5.2.2 code . . . 16

6 Conclusão 18

Chapter 1

Algoritmo

1.1 Algoritmo

1.2 Opção de algoritmo "Vetor"

A escolha de como os dados são armazenados no programa em runtime, em contexto com o problema a resolver, um limite de memória a alocar não seria aceitável, sentimos então a necessidade da implementação de um algoritmo em que o armazenamento de dados é dinamicamente alocado conforme as necessidades de utilização de Veículos e Ordens de utilização, o que permite ao utilizador o uso de um programa otimizado e capaz de se autodimensionar em memoria conforme as necessidadas apresentatas, o limite será então a quantidade de memória ram do sistema, ou 2^{64} visto que na estrutura do vetor dinamico em que são guardados endereços de memória (*pointers) é usado size_t (tipo de dados positivo que fica com o tamanho da arquitetura do cpu) para indexar o array, o maior elemento que um sistema comum de 64 bits consegue guardar numa variavel.

1.2.1 Vantagens de um array dinâmico

- quantidade de elementos ilimitados.
- reduzido uso de memória em comparação a listas ligadas, o que permite a alocação de bastante informação.
- 'indexing', é possivel aceder aos elementos do array através de um index, sem necessitar iterações adicionais.
- dealocação e clone é possivel com apenas duas operações.

1.2.2 Desvantagens de um array dinâmico

- inserção e remoção á cabeça.
- divisão ou junção de elementos.
- a posição na memória não é constante, o que força em algumas situações o uso de '**void' na implementação, o que poderia permitir também o cast de apontadores genericos '*void'.

1.3 Estruturas de dados

Para a implementação do vetor será necessário o uso de estrutas de dados bem defenidas que detêm varios valores associados a um veículo ou ordem, o que possibilita guardar os seus endereços de memoria num array que não dispõe de limite em runtime.

```c
typedef struct Vehicle {
  char *id;
  char *type;
  float price;
  uint32_t autonomy;
  bool active; // control when removing, 1 important byte!
} Vehicle;

typedef struct Order {
  size_t id;
  size_t nif; // 2**32 is not enough for the world's population
  // Vehicle *v_id; //! LOGIC ERROR
  char *v_id;
  uint32_t time;
  uint32_t distance;
} Order;
```

1.3.1 Veículos

Nesta estrutura guardamos a informação relativa a um veículo, dois apontadores para strings que representam o Id e tipo de veículo, Id este que necessita ser único e garantimos no nosso programa que isso se verifica, guardamos também, um valor real (float) relativo ao preço de utilização, um valor inteiro de 32bits que representa a autonomia do veículo, e um byte de controlo para indicar se o veículo se encontra atualmente ativo ,valores que foram devidamente validados "!TODO",

1.3.2 Pedidos de utilização

Nesta estrutura guardamos a informação relativa a um pedido de utilização de um veículo, dois valores inteiros positivos do tamanho da arquitetura do CPU(64bits) que representam o id (automaticamente atribuído) e um nif com mais de 8 digitos, intensionava-mos guardar também um apontador para um veículo já alocado no programa, competente para a finalização da ordem, valor na memória este que não seria estático, como se trata de um vetor, apontadores para os seus elementos variam sempre que este é ordenado, concluimos então que uma lista ligada seria mais adequada para guardar os veículos ainda que, fosse necessária mais memória ram, um apontador constante para um veiculo iria evitar bastantes iterações sempre que fosse necessária informações do veículo relacionado, em vez de guardar um apontador para uma estrutura veículo guardamos um apontador para uma string(Id de veículo), mas este é um algoritmo relativamente ineficiente, posteriormente iremos alterar o código associado aos vetores para um vetor do tipo genérico void **, e usar uma lista ligada para guardar os veículos em memória.

1.4 implementação do Vetor

```c
#define V_ALLOC 64
// A contiguos growable array type "vector"
// +------+----------+---------+
// | size | capacity | data[]  |
// +------+----------+---------+
//                   ^
//                   | given pointer

typedef struct vehicle_vec {
  Vehicle *data;
```

```
  size_t len, capacity;
} Vehicles;

typedef struct order_vec {
  Order *data;
  size_t len, capacity;
} Orders;
```

- size_t len, capacity, aqui guardamos informação essencial para o autodimensionamento do array em runtime, os elementos inicializados e relevantes no contexto são sempre os elementos entre 0 e vetor->len -1, a capacidade é a quantidade alocada até ao momento.
- usamos a função malloc que retorna um apontador para a área alocada na memória caso a alocação seja permitida pelo sistema operativo, também consideramos usar calloc mas a memória inicializada não nos iria trazer grande vantagem e iria usar mais ciclos de processamentos desnecessários.
- o array de apontadores (*data) com cast (Order ou Vehicle) do tipo não genérico permite guardar os vários apontadores para uma estrutura de dados completa em contexto de uso, também verificamos que NULLs não causam comportamento inesperado no programa, a realocação é feita sempre que o vetor está a 100% de capacidade ou 25% em que seria aumentado e diminuido respetivamente.

```
#include "vec.h"

Orders *vec_orders_new() {
  Orders *v = (Orders *)malloc(sizeof(Orders));
  if (v == NULL) {
    LOG_FATAL("Not able to allocate memory for Orders", 12);
  }
  v->len = 0;
  v->capacity = V_ALLOC;
  v->data = (Order *)malloc(sizeof(Order) * v->capacity);
  if (v->data == NULL) {
    LOG_FATAL("Not able to allocate memory for Orders", 12);
  }
  return v;
}

static inline void vec_orders_expand(Orders *v) {
  assert(v);
  size_t new_capacity = 2 * v->capacity;
  Order *new_array = (Order *)malloc(sizeof(Order) * new_capacity);
  if (new_array == NULL) {
    LOG_FATAL("Not able to expand memory for Orders", 12);
  }
  for (size_t i = 0; i < v->len; i++) {
    new_array[i] = v->data[i];
  }
  free(v->data);
  v->data = new_array;
  v->capacity = new_capacity;
}

static inline void vec_orders_halve(Orders *v) {
  assert(v);
  size_t new_capacity = v->capacity / 2;
  Order *new_array = (Order *)malloc(sizeof(Order) * new_capacity);
```

```
  if (new_array == NULL) {
    LOG_FATAL("Not able to halve memory for Orders", 12);
  }
  for (size_t i = 0; i < v->len; i++) {
    new_array[i] = v->data[i];
  }
  free(v->data);
  v->data = new_array;
  v->capacity = new_capacity;
  v->len = v->len < new_capacity ? v->len : new_capacity;
}
bool vec_orders_is_empty(Orders *v) {
  assert(v);
  return v->len == 0;
}

size_t vec_orders_len(Orders *v) {
  assert(v);
  return v->len;
}

Order *vec_orders_get(Orders *v, size_t idx) {
  assert(v);
  if (idx >= v->len) {
    LOG_FATAL("Index does not exist", 1);
  }
  return &v->data[idx];
}

void vec_orders_push(Orders *v, Order *value) {
  assert(v);
  if (v->len == v->capacity) {
    vec_orders_expand(v);
  }
  v->data[v->len++] = *value;
}

void vec_orders_change_at(Orders *v, size_t i, Order *value) {
  assert(v);
  if (i >= v->len) {
    LOG_FATAL("Index does not exist", 1);
  }
  v->data[i] = *value;
}

void vec_orders_push_at(Orders *v, size_t i, Order *value) {
  assert(v);
  if (i >= v->len) {
    LOG_FATAL("Index does not exist", 1);
  }
  if (v->len == v->capacity) {
    vec_orders_expand(v);
  }
  for (size_t x = v->len; x > i; x--) {
    v->data[x] = v->data[x - 1];
  }
  v->data[i] = *value;
  v->len++;
}

void vec_orders_rm_at(Orders *v, size_t i) {
  assert(v);
  if (i >= v->len) {
```

```
    LOG_FATAL("Index does not exist", 1);
  }
  for (size_t x = i + 1; x < v->len; x++) {
    v->data[x - 1] = v->data[x];
  }
  v->len--;
  if (v->len < v->capacity / 4) {
    vec_orders_halve(v);
  }
}

void vec_orders_destroy(Orders *v) {
  assert(v);
  free(v->data);
  free(v);
}

void vec_orders_reset(Orders *v) {
  assert(v);
  vec_orders_destroy(v);
  v = vec_orders_new();
}
```

Chapter 2

makefile

Na parte inicial do projeto usamos um simples makefile para facilitar a compilação e o linking do programa, definimos também uma condição para usar o clang em processadores arm64 e e gcc para x86_64, usamos a flag -03 e -g para compilar o projeto em release ou debug mode respetivamente, definimos os ficheiros source e os objetos relacionados na específica linguagem do make o que permite um compilação fácil e objetiva.

```
EXEC  = $(shell basename $$(pwd))
ARCH := $(shell uname -p)
#MODIFY IF NEDDED
ifeq ($(ARCH), x86_64)
        CC = gcc
else
        CC = clang
endif
#replace -03 with -g to debug
CFLAGS= -c                \
        -03               \
        -W                \
        -Wall             \
        -pedantic
SRC     = $(wildcard *.c)
OBJ     = $(SRC:.c=.o)
LDFL    = #libs like math...

all: $(EXEC)

${EXEC}: $(OBJ)
        $(CC) -o $@ $^ $(LDFL)

%.o: %.c
        $(CC) -o $@ $< $(CFLAGS)

.PHONY: clean mrproper

clean:
        @rm -rf *.o

mrproper: clean
        @rm -rf $(EXEC)
```

Chapter 3

Validação de dados

3.1 síntese

Todos os dados são validados antes de serem guardados em memória(vec push), até na leitura de ficheiros verificamos alguma da sua integridade.

- validamos que Id's de veículos não se repetem.
- o tipo de veículos é livre para o utilizador escolher.
- o preço e autonomia de veículos têm limites também definidos.
- o id é automatimente atribuído por ordem de pedido.
- o nif necessita de ter 8+ digitos, usamos size_t(64bits) porque 32 não chegam para a população mundial.
- distância e tempo também são devidamente verificados.

3.2 código

Docs!! Static definitions could hide the validation proccess from other libs, the program is insecure since pointers are given with memory that's not readonly, so "full" control is granted, data validation should be managed differently in a more secure environment, given pointers to other libs should point only to allocated temporary structs, not "all" data the program has, if thats nedded then the const modifier would have to be considered in some places to avoid security issues. Clones and read only memory managed at some degree of control, as the project grows some areas are not meant to be accessable from others.

```
size_t assign_oid(Orders *v) {
  // return (v->len)+1; // v->len is dynamic so..., this doesn't work at
     all
  // so we have to loop through the vec and find the biggest value
  size_t m = 0;
  for (size_t i = 0; i < v->len; i++) {
    if (((&v->data[i])->id) > m)
      m = (&v->data[i])->id;
  }
  return m + 1;
}

bool vehicle_id_exists(Vehicles *v, const char *id) {
  for (size_t i = 0; i < v->len; i++) {
    if ((strcmp(id, (&v->data[i])->id)) == 0)
```

```
      return true;
  }
  return false;
}

bool order_id_exists(Orders *v, const size_t id) {
  for (size_t i = 0; i < v->len; i++) {
    if (id == (&v->data[i])->id)
      return true;
  }
  return false;
}

static inline uint8_t invalidate_price(float *price) {
  return (*price > 0) && (*price < 1000000) ? 0 : 1;
}

static inline uint8_t invalidate_autonomy(uint32_t *autonomy) {
  return (*autonomy != 0 && *autonomy <= 10000) ? 0 : 1;
}

static inline uint8_t invalidate_nif(size_t *nif) {
  return (*nif >= 10000000) ? 0 : 1;
}

static inline uint8_t invalidate_time(uint32_t *time) {
  return (*time <= 10000 && *time != 0) ? 0 : 1;
}

static inline uint8_t invalidate_distance(uint32_t *distance) {
  return (*distance <= 10000 && *distance != 0) ? 0 : 1;
}

static inline size_t calculate_dst(Vehicle *v_id, Orders *o) {
  if (v_id->active == false)
    return 0;
  size_t distance_needed = 0;
  for (size_t i = 0; i < o->len; i++) {
    if (v_id == (&o->data[i])->v_id)
      distance_needed += (&o->data[i])->distance;
  }
  return distance_needed;
}

static inline Vehicle *search_vehicle_by_type(Vehicles *v, const char *
    type,
                                              size_t *pos) {
  for (size_t i = *pos; i < v->len; i++) {
    if (((strcmp(type, (&v->data[i])->type)) == 0) && (i >= *pos)) {
      *pos = i + 1;
      return &v->data[i];
    }
  }
  return NULL;
}

Vehicle *assign_vid(Vehicles *v, Orders *o, char *v_id_str,
                    const uint32_t distance) {
  Vehicle *v_id = search_vehicle_by_id(v, v_id_str);
  uint32_t distance_needed = distance;
  if ((distance_needed += calculate_dst(v_id, o)) <= v_id->autonomy) {
    return v_id;
  }
```

```
  // try find another vehicle with the same type
  Vehicle *temp = v_id;
  for (size_t i = 0; v_id != NULL;
       v_id = search_vehicle_by_type(v, v_id->type, &i)) {
    if (temp == v_id) // dont check the same vehicle
      continue;
    distance_needed = distance; // reset variable
    if ((distance_needed += calculate_dst(v_id, o)) <= v_id->autonomy)
      return v_id;
  }
  return NULL;
}

bool invalidate_order(Vehicles *v, Orders *o, Order *oid) {
  if (oid->v_id == NULL) {
    free(oid);
    return true;
  }
  if (oid->id < assign_oid(o) || invalidate_nif(&oid->nif) ||
      invalidate_time(&oid->time) || invalidate_distance(&oid->distance)
    ) {
    free(oid);
    return true;
  }
  Vehicle *vid = assign_vid(v, o, oid->v_id, oid->distance);
  if (vid == NULL) {
    free(oid);
    return true;
  }
  oid->v_id = strdup(vid->id);
  if (vid->active == false)
    vid->active = true;
  return false;
}

bool invalidate_vehicle(Vehicles *v, Vehicle *vid) {
  if (vehicle_id_exists(v, vid->id) || invalidate_price(&vid->price) ||
      invalidate_autonomy(&vid->autonomy)) {
    free(vid);
    return true;
  }
  return false;
}
```

3.3 interpretação

- Na função assign_vid respondemos automaticamente ao tópico 10 onde se pretende que o veículo que o utilizador deseja seja validado se possível, procuramos então validar esses dados logo no instatne de pedido de ordem e invalidar, caso não haja, veículos disponiveis do tipo desejado.

- As funções 'invalidate_vehicle' e 'invalidate_order', recebem o apontador de uma estrutura alocada pronta para ser adicionada ao vetor caso passe todos as verifica;oes e seja validada.

Chapter 4

Leitura de ficheiros

4.1 funções

Para abrirmos o ficheiro no código, usamos fopen com o parâmetro r e w+ para leitura e escrita, respetivamente, também verificamos que estes apontadores retornado (FILE *) não são NULLs, o que iria crashar o programa.

Usamos as funções sscanf e fprintf para leitura e escrita de ficheiros, respetivamente. São lidos valores separados por tab '.tsv' "tab separated file" linha por linha enquanto a função getline retorna um valor diferente de -1 para a variavel read (ssize_t(size_t adequado para a representação de alguns números negativos)), na proxima operação verificamos que a função sscanf retorna exatamente 4 valores lidos caso contrário os dados sejam invalidados, invalidamos por completo a leitura de ficheiros e informamos o utitlizador.

4.2 codigo

```
static inline uint8_t read_vehicles(Vehicles *v) {
  FILE *fp;
  if ((fp = fopen("./data/vehicles.tsv", "r")) == NULL) {
      LOG_ERRNO(2);
      LOG_WARN_R("Vehicles not read!")
     return 2;
  }
  char *line = NULL;
  size_t len;
  ssize_t read;

  char *id = malloc(sizeof(char) * VEHICLE_ID_MAX_CHARS);
  char *type = malloc(sizeof(char) * VEHICLE_TYPE_MAX_CHARS);
  float price = 0;
  unsigned int autonomy = 0;

  while ((read = getline(&line, &len, fp)) != -1) {
    if ((sscanf(line, "%s\t%s\t%f\t%u", id, type, &price, &autonomy)) !=
     4) {
      LOGF_ERR("Corrupted Vehicles file, not read!");
       vec_vehicles_reset(v);
       free(line);
       free(id);
       free(type);
       fclose(fp);
     return 1;
    }
    Vehicle *data = vehicle_build(id, type, price, autonomy);
    if (invalidate_vehicle(v, data)) {
```

```
      LOGF_ERR("Corrupted Vehicles file, not read!");
      vec_vehicles_reset(v);
      free(line);
      free(id);
      free(type);
      fclose(fp);
     return 1;
    }
    vec_vehicles_push(v, data);
  }

  if (line) {
    free(line);
  }
     free(id);
     free(type);
     fclose(fp);
    return 0;
}

static inline uint8_t write_vehicles(Vehicles *v) {
  FILE *fp;
  if ((fp = fopen("./data/vehicles.tsv", "w+")) == NULL) {
     LOG_ERRNO(2);
     LOG_WARN_R("Vehicles not written!")
    return 2;
  }
  for (size_t i = 0; i < v->len; i++) {
    fprintf(fp, "%s\t%s\t%f\t%u\n", (&v->data[i])->id, (&v->data[i])->
    type,
    (&v->data[i])->price, (&v->data[i])->autonomy);
  }
     fclose(fp);
    return 0;
}

  static inline uint8_t read_orders(Vehicles *v, Orders *o) {
  FILE *fp;
  if ((fp = fopen("./data/orders.tsv", "r")) == NULL) {
     LOG_ERRNO(2);
     LOG_WARN_R("Orders not read!");
    return 2;
  }
  char *line = NULL;
  size_t len;
  ssize_t read;

  size_t id = 0;
  size_t nif = 0;
  char *v_id_str = malloc(sizeof(char) * VEHICLE_ID_MAX_CHARS);
  unsigned int time = 0;
  unsigned int distance = 0;

  while ((read = getline(&line, &len, fp)) != -1) {
    if (sscanf(line, "%lu\t%lu\t%s\t%u\t%u", &id, &nif, v_id_str, &time,
               &distance) != 5) {
      LOGF_ERR("Corrupted Orders file, not read!");
      vec_orders_reset(o);
      free(line);
      free(v_id_str);
      fclose(fp);
      return 1;
    }
```

```
    Order *data = order_build(id, nif, v_id_str, time, distance);
    if (invalidate_order(v, o, data)) {
      LOGF_ERR("Corrupted Orders file, not read!");
      vec_orders_reset(o);
      free(line);
      free(v_id_str);
      fclose(fp);
      return 1;
    }
    vec_orders_push(o, data);
  }

  if (line) {
    free(line);
  }
    free(v_id_str);
    fclose(fp);
   return 0;
}

    static inline uint8_t write_orders(Orders *v) {
    FILE *fp;
    if ((fp = fopen("./data/orders.tsv", "w+")) == NULL) {
    LOG_ERRNO(2);
    LOG_WARN_R("Orders not written!");
   return 2;
  }
  for (size_t i = 0; i < v->len; i++) {
    fprintf(fp, "%lu\t%lu\t%s\t%u\t%u\n", (&v->data[i])->id, (&v->data[i
    ])->nif,
    (&v->data[i])->v_id, (&v->data[i])->time, (&v->data[i])->distance);
  }
    fclose(fp);
   return 0;
}

    uint8_t read_data_err(Vehicles *v, Orders *o) {
    if (read_vehicles(v) | read_orders(v, o)) {
    return 1;
  }
   return 0;
}

    uint8_t write_data_err(Vehicles *v, Orders *o) {
    if (write_vehicles(v) | write_orders(o)) {
   return 1;
  }
  return 0;
```

Chapter 5

Utilitarios

5.1 Logger

Criamos também as nossas próprias funções de logging e ainda que bastante simples ajudam a entender e deixam registado o comportamento do programa/utilizador

5.1.1 header

```
#ifndef __LOG_H__
#define __LOG_H__
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#ifndef L_PATH
#define L_PATH "./logs/"
#endif // L_PATH

typedef struct Log { // never constructed, no time! \:
  FILE *fp;          // we wanted the file opened while the program is
    running.
  struct tm *time;
} Log;

#ifndef L_ERR
#define L_ERR "|ERROR|\t"   // error reports
#define L_WARN "|WARN |\t"  // warning messages
#define L_INFO "|INFO |\t"  // general info reported
#define L_CMMND "|CMMND|\t" // commands executed in runtime
#define L_SUCMD "|SUCMD|\t" // as super user
#define L_FATAL "|FATAL|\t" // exiting errors reported
#define L_PANIC "|PANIC|\t" // when a FATAL log is not reported, logged
    latter
#define L_DEBUG "|DEBUG|\t" // used for development, need callback
    functions btw
#define L_TEST "|TEST |\t"  // production tests
#define L_TRACE "|TRACE|\t" // detailed steps of an operation
#endif                      // L_ERR // log to files

#ifndef LC_ERR
#define LC_ERR "\033[91m|ERROR|\t"
#define LC_FATAL "\033[91m|FATAL|\t"
#define LC_WARN "\033[93m|WARN |\t"
#define LC_WARN_R "\033[91m|WARN |\t"
#define LC_INFO "\033[92m|INFO |\t"
#define LC_RESET "\033[0m"
```

```
#endif // LC_ERR // log to terminal

void log_to_file(const char *str);
void log_errno_to_file(const uint8_t err);

#ifndef LOG_ERR
#define LOG_ERRNO(ERRNO)
          \
  fprintf(stderr, LC_ERR "%d : %s!\n" LC_RESET, ERRNO, strerror(ERRNO));
          \
  log_errno_to_file(ERRNO);

// doesn't log to file
#define LOG_ERR(STR) fprintf(stderr, LC_ERR "%s" LC_RESET "\n", STR)

#define LOGF_ERR(STR)
          \
  fprintf(stderr, LC_ERR "%s" LC_RESET "\n", STR);
          \
  log_to_file(L_ERR STR);
#endif // LOR_ERR

#ifndef LOG_INFO
#define LOG_INFO(STR)
          \
  printf(LC_INFO "%s" LC_RESET "\n", STR);
          \
  log_to_file(L_INFO STR)
#define LOG_INFO_NW(STR)
          \
  printf(LC_INFO "%s" LC_RESET "\n", STR);
          \
//! TODO
#endif // LOG_INFO

#ifndef LOG_WARN
#define LOG_WARN(STR)
          \
  printf(LC_WARN "%s" LC_RESET "\n", STR);
          \
  log_to_file(L_WARN STR);

#define LOG_WARN_R(STR)
          \
  printf(LC_WARN_R "%s" LC_RESET "\n", STR);
          \
  log_to_file(L_WARN STR);
//! TODO
#endif // LOG_WARN

#ifndef LOG_FATAL
#define LOG_ERRNO_EXIT(ERRNO)
          \
  log_errno_to_file(ERRNO);
          \
  fprintf(stderr, LC_FATAL "exit(%d) : %s!" LC_RESET, ERRNO, strerror(
    ERRNO)); \
  exit(ERRNO);

#define LOG_FATAL(STR, ERRNO)
          \
  LOG_ERR(STR);
          \
```

```
  LOG_ERRNO_EXIT(ERRNO)
#endif // LOG_FATAL

#endif // __LOG_H__
```

5.2 Teste

Também temos uma função de teste automáticos e benchmarks relacionados

5.2.1 header

```
#define __TEST_H__
//! Not used in production, move to src dir, call all_tests(), and
    compile
//! the project with -g flag, not the best approach to automated testing
     but
//! they're enough for now.

#include "vec.h"
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define _tstart puts("------Starting tests!------");

#define _tfail(STR) printf("!!!-TEST-FAILED--->%s\n", STR);
#define _tpass(STR) printf("!!!-TEST-PASSED--->%s\n", STR);

// benchmarks
#define _tbench_start(STR)
          \
  printf("!===STARTING BENCHS FOR %s ===!\n", STR);
          \
  float start_t = (float)clock() / CLOCKS_PER_SEC;

#define _tbench_stop
          \
  float end_t = (float)clock() / CLOCKS_PER_SEC;
          \
  float bench = end_t - start_t;

#define _tbench_print(STR) printf("!===BENCH FOR %s===>%fseg\n", STR,
    bench);

#define _assert_fn(TEST, MSG)
          \
  if (!TEST) {
          \
    LOG_ERR(MSG);
          \
    _tfail(MSG);
          \
  }
          \
  _tpass(MSG);

// exits on failure
```

```
#define _assert(TEST, MSG)
                \
  if (!TEST) {
                \
    LOG_ERR(MSG);
                \
    _tfail(MSG);
                \
    exit(-1);
                \
  }
                \
  _tpass(MSG);

#endif // __TEST_H__
```

5.2.2 code

```
#include "test.h"

static void test_vehicles_vec() {
  _tstart;
  _tbench_start("Vehicles vec");
  Vehicles *v = vec_vehicles_new();
  vec_vehicles_push(v, vehicle_build("M_21", "carro", 3.5, 2));
  Vehicle *v_id = v->data;
  _assert((v->capacity == 64), "vec capacity");
  _assert((v->len == 1), "vec len");
  _assert((v_id != NULL), "vec push");
  _assert(strcmp(v_id->id, "M_21") == 0, "vehicle id");
  _assert(strcmp(v_id->type, "carro") == 0, "vehicle type");
  _assert((v_id->price == 3.5), "vehicle price");
  _assert((v_id->autonomy == 2), "vehicle autonomy");
  for (size_t i = 0; i < 640000; i++) {
    vec_vehicles_push(v, vehicle_build("M_21", "carro", 3.5, 2));
  }
  _assert(((v->capacity == 1048576) && (v->len == 640001)), "vec expand"
    );
  _assert(v_id == (&v->data[0]), "vec memory access");
  // free strings
  for (size_t i = 0; i < v->len; i++) {
    if ((&v->data[i])->id != NULL) {
      free((&v->data[i])->id);
      (&v->data[i])->id = NULL;
    }
    if ((&v->data[i])->type != NULL) {
      free((&v->data[i])->type);
      (&v->data[i])->type = NULL;
    }
  }
  _assert(((&v->data[32])->id == NULL), "string id free");
  _assert(((&v->data[32])->type == NULL), "string type free");
  free(v->data);
  free(v);
}

//! TODO

static void bench_vehicles_vec() {
  _tbench_start("Vehicles vec");
  Vehicles *v = vec_vehicles_new();
```

```
  for (size_t i = 0; i < 100000000; i++) {
    vec_vehicles_push(v, vehicle_build("M_12", "carro", 3.5, 2));
  }
  _tbench_stop;
  _tbench_print("vehicles vec");
  vec_vehicles_destroy(v);
}

static void all_benchs() {
  bench_vehicles_vec();
  //
}

void all_tests() {
  test_vehicles_vec();
  all_benchs();
}
```

Chapter 6

Conclusão

Com este trabalho concluimos que este projeto foi bastante importante para a nossa aprendizagem e para elevar o nosso trabalho em grupo. Aprendemos a trabalhar em LaTex o que nos vai ser muito útil para o futuro. Gostariamos de agradecer aos docentes acima referidos por nos incentivar a requirir o trabalho.