 ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO	Tipo de Prova Mini Teste 2 (Avaliação Contínua)	Ano letivo 2019/2020	Data 31/01/2020
	Curso Licenciatura em Engenharia Informática Licenciatura em Segurança Informática em Redes de Computadores	Hora 10:00	
	Unidade Curricular Fundamentos de Programação	Duração 2:00 horas	

Observações:

- Pode trocar a ordem das questões, desde que as identifique convenientemente.
- É permitida a utilização da biblioteca de input de dados (API_Leitura.h) abordada nas aulas, disponibilizada em anexo.
- Todos os exercícios devem ser resolvidos utilizando a linguagem de programação C.
- Qualquer tentativa de fraude implica a anulação do exame.


Na resolução desta prova considere as estruturas de dados que são apresentadas de seguida e que modelam de forma simplista o funcionamento das meetups organizadas pela Data Science Portugal (DSPT). A DSPT é uma associação que tem como objetivo promover a disseminação da área de Data Science e, nesse sentido, organiza meetups (reuniões) frequentes para os seus membros. Nestas reuniões, em que apenas os membros podem participar, são convidados no máximo 3 speakers para fazerem apresentações para a comunidade. Para além do nome do speaker, a DSPT guarda o título, o resumo (abstract) e a duração (em minutos) da sua apresentação. Para além das listas dos speakers e dos identificadores dos participantes, para cada meetup, é registado o seu número (sequencial), o seu título, a cidade em que ocorre, bem como o número de speakers, participantes e cervejas consumidas.

```
#define MAX_STR 100
#define MAX_SPEAKERS 3

struct member{
    unsigned int id;
    char name[MAX_STR + 1], city[MAX_STR + 1];
};

struct speaker{
    char name[MAX_STR + 1], title[MAX_STR + 1], abstract[MAX_STR + 1];
    unsigned short duration_min;
};

struct meetup{
    unsigned short number, year;
    char title[MAX_STR + 1], city[MAX_STR + 1];
    struct speaker speakers[MAX_SPEAKERS];
    unsigned int* participant_ids;
    unsigned short n_speakers, n_participants, n_beers;
};
```

 <div>ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO</div>	Tipo de Prova Mini Teste 2 (Avaliação Contínua)	Ano letivo 2019/2020	Data 31/01/2020
	Curso Licenciatura em Engenharia Informática Licenciatura em Segurança Informática em Redes de Computadores		Hora 10:00
	Unidade Curricular Fundamentos de Programação		Duração 2:00 horas

1. Implemente a função **free_n_meetups**, cuja assinatura é apresentada abaixo. Esta função é utilizada quando algo falha durante a inicialização de uma lista de meetups, tornando-se necessário libertar toda a memória entretanto alocada e encerrar o programa.

(2.5V)

(15 min)


```
/**
 * Função que liberta toda a memória alocada de uma lista de meetups que
 * tenha sido corretamente inicializada apenas até à posição n
 *
 * @param meetups apontador duplo para a lista de meetups a libertar
 * @param n_meetups apontador para o número de meetups na lista
 * @param max_meetups apontador para a capacidade máxima da lista de meetups
 * @param n posição na lista de meetups na qual algo falhou
 */
void free_n_meetups(struct meetup** meetups, unsigned int *n_meetups,
    unsigned int *max_meetups, unsigned int n);
```
2. Implemente a função **initialize_meetup_list**, cuja assinatura é apresentada abaixo. Esta função inicializa uma lista de meetups, deixando-a pronta para receber informação. Tenha em consideração que:

(5V)

(30 min)

 - As strings devem ser inicializadas com a string vazia;
 - A lista de participantes deve ser inicializada para 10 participantes;
 - O ano deve ser 2020;
 - Os números das meetups devem ser sequenciais e crescentes, começando em 1;
 - Variáveis numéricas como a `duration_min` ou a `n_beers` devem ser inicializadas a 0;
 - Se, a meio da execução, algo falhar, a função deve libertar toda a memória entretanto alocada e devolver o valor 0. Se assim o entender pode utilizar a função **free_n_meetups** (descrita no ponto 1) mesmo que não a tenha implementado.

```
/**
 * Função que inicializa uma lista de meetups
 *
 * @param meetups apontador duplo para a lista a inicializar
 * @param n_meetups apontador para o número de meetups na lista
 * @param max_meetups apontador para a capacidade máxima da lista
 * @param n tamanho pretendido para a lista de meetups
 * @return 1 em caso de sucesso, 0 caso contrário
 */
int initialize_meetup_list(struct meetup** meetups, unsigned int *n_meetups,
    unsigned int *max_meetups, unsigned int n);
```

 <div>ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO</div>	Tipo de Prova Mini Teste 2 (Avaliação Contínua)	Ano letivo 2019/2020	Data 31/01/2020
	Curso Licenciatura em Engenharia Informática Licenciatura em Segurança Informática em Redes de Computadores		Hora 10:00
	Unidade Curricular Fundamentos de Programação		Duração 2:00 horas

3. Implemente a função **beers_year**, cuja assinatura se apresenta abaixo. Esta função calcula o número médio de cervejas consumidas por participante num determinado ano. Verifique se existe algum caso em que esta função possa falhar e, em caso afirmativo, implemente a função de forma a que não falhe e devolva o valor -1 nessa situação. Assuma que os argumentos da função estão devidamente inicializados.
- (2.5V)
- (15 min)

```
/**
 * Função que calcula o número médio de cervejas por participante consumidas
 * num determinado ano
 *
 * @param meetups apontador para a lista de meetups
 * @param n_meetups número de meetups na lista
 * @param year ano para o qual estamos a calcular a média
 * @return
 */
double beers_year(struct meetup* meetups, unsigned int n_meetups,
                  unsigned int year);
```

4. Considere o programa que se apresenta abaixo que lê uma lista de membros e os coloca na respetiva lista, e inicializa a lista de meetups utilizando a função descrita no ponto 2. De seguida, entra num menu que inclui as seguintes opções:
- (3V)
- (15min)
- Gestão de membros (adicionar, pesquisar, remover, editar)
 - Gestão de meetups (adicionar, pesquisar, editar)
 - Sair do menu

Considere ainda que a função **read_members** lê os dados de uma qualquer fonte e nunca falha.

O programa contém 3 erros. Identifique-os e escreva na folha de resposta uma nova versão do programa com os erros corrigidos. Cada erro corrigido vale 1 valor, cada alteração que prejudique o funcionamento do programa desconta 0.5 valores.

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


    struct member* members;
    struct meetup* meetups;
    unsigned int n_members, n_meetups, max_members, max_meetups;

    //ler a lista de membros
    read_members(&members, &n_members, &max_members);

    int res = initialize_meetup_list(&meetups, &n_meetups, &max_meetups, 10);
    if (res == 1){
        //entra no ciclo do menu principal
        menu_principal(&members, n_members, max_members, &meetups, n_meetups, max_meetups);
    }

    free_n_meetups(&meetups, &n_meetups, &max_meetups, max_meetups);

    return (EXIT_SUCCESS);
}
```

 <small>ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO</small>	Tipo de Prova Mini Teste 2 (Avaliação Contínua)	Ano letivo 2019/2020	Data 31/01/2020
	Curso Licenciatura em Engenharia Informática Licenciatura em Segurança Informática em Redes de Computadores	Hora 10:00	
	Unidade Curricular Fundamentos de Programação	Duração 2:00 horas	


5. Implemente a função `get_first_participant`, cuja assinatura é apresentada abaixo. Esta função deve devolver o endereço de memória do primeiro membro inscrito num meetup com um número dado como argumento. Assuma que todas as listas estão corretamente preenchidas e que todos os identificadores de membros nos meetups existem efetivamente na lista de membros. A função deve devolver NULL em caso de insucesso.

(3.5V)
(20 min)

```
/**
 * Devolve um apontador para o primeiro membro inscrito num meetup com um
 * dado número.
 *
 * @param meetups apontador para a lista de meetups
 * @param n_meetups número de meetups na lista
 * @param members apontador para a lista de membros
 * @param n_members número de membros na lista
 * @param meetup_number número do meetup a pesquisar
 * @return um apontador para o membro ou NULL
 */
struct member* get_first_participant(struct meetup* meetups,
                                     unsigned int n_meetups, struct member* members, unsigned int n_members,
                                     unsigned int meetup_number);
```

6. Considere agora que a DSPT pretendia acrescentar funcionalidades ao seu sistema, nomeadamente:
- Pretende-se registar as presenças em cada meetup para permitir saber que membros estiveram presentes e que membros faltaram (**1 Valor**);
 - A DSPT pretende passar a fazer a gestão independente de uma lista de speakers (tal como faz de membros), sendo que cada speaker tem uma lista de talks (palestras). Isto permitirá mais facilmente saber todas as talks dadas por um determinado speaker (neste momento é necessário percorrer a lista de meetups e respetivos speakers, pesquisando pelo nome). Reformule as estruturas para que exista o conceito de talk (palestra), caracterizado pelo título, abstract e duração (tal como na versão atual no speaker), e que, para além disso, tenha também um identificador numérico inteiro. Continua a existir o limite atual de talks por meetup (3) mas cada speaker pode ter um número indefinido de talks. (**2.5 valores**).

(3.5V)
(20 min)

 ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO	Tipo de Prova Mini Teste 2 (Avaliação Contínua)	Ano letivo 2019/2020	Data 31/01/2020
	Curso Licenciatura em Engenharia Informática Licenciatura em Segurança Informática em Redes de Computadores	Hora 10:00	
	Unidade Curricular Fundamentos de Programação	Duração 2:00 horas	

Anexo

Conteúdo do header file API_Leitura.h:

```
void readShort(short *const value, const short minValue,
               const short maxValue, char const* const message);

void readInt(int *const value, const int minValue,
             const int maxValue, char const* const message);

void readLong(long *const value, const long minValue,
              const long maxValue, char const* const message);

void readFloat(float *const value, const float minValue,
               const float maxValue, char const* const message);

void readDouble(double *const value, const double minValue,
                const double maxValue, char const* const message);

void readChar(char *const value, char const* const message);

bool readString(char *const value, const unsigned int size,
                char const* const message);

void readBool(bool *const value, char const* const message);
```

Conteúdo parcial do header file string.h:

```
size_t strlen (const char *s)
char * strcpy (char *restrict to, const char *restrict from)
int strcmp (const char *s1, const char *s2)
```