

Ficha de Trabalho n.º 4

Objectivos: Estruturas dinâmicas - **Filas de Espera.**

Os exercícios propostos nesta ficha visam criar um programa que facilite a gestão de um serviço de lavagens de viaturas segundo uma lógica FIFO (First In First Out).

1. Defina uma estrutura de dados adequada para manter em memória RAM os seguintes dados relativos a uma determinada viatura:
 - a) Matrícula (8 caracteres);
 - b) Proprietário (30 caracteres);
 - c) Marca (15 caracteres);
 - d) Cor (15 caracteres);
 - e) Tempo de entrada (time_t);
 - f) Tempo de saída (time_t);
 - g) Tempo de espera e lavagem em segundos (inteiro).
2. Defina estruturas de dados adequadas para manter em memória uma fila de viaturas.
3. Escreva uma função que faça a leitura dos dados relativos a uma viatura que chega ao serviço de lavagens, procedendo também ao registo do tempo de entrada.
4. Escreva uma função que mostre os dados relativos a uma dada viatura em espera.
5. Escreva uma função que liste todas as viaturas em espera, mostrando os seus dados.
6. Escreva uma função que calcule o tempo de lavagem da viatura em segundos.
7. Escreva uma função que permita saber quantas viaturas estão ainda à espera da lavagem.
8. Escreva uma função que permita inserir uma viatura na fila de espera.
9. Escreva uma função que permita retirar uma viatura da fila de espera, mas que antes preencha o tempo de saída e calcule o tempo de lavagem para mostrar ao utilizador.
10. Elabore uma função que grave os dados das viaturas na fila num ficheiro "fila.dat".
11. Elabore uma função que permita ler os dados no ficheiro "fila.dat" e adicione as respectivas viaturas à fila de espera.
12. Elabore uma função que permita limpar a fila (eliminar todas as viaturas existentes).
13. Escreva uma função MENU () que oriente o utilizador do programa na escolha das diferentes funcionalidades que lhe são proporcionadas.
14. Escreva um programa que integre as funções descritas nos números anteriores, acrescentando eventualmente algumas funções necessárias e não solicitadas anteriormente.

APÊNDICE

<time.h> - Time and Date functions

CLOCKS_PER_SEC The number of [clock_t](#) units per second.

NULL Null pointer constant.

clock_t An arithmetic type elapsed processor representing time.

time_t An arithmetic type representing calendar time.

struct tm Represents the components of calendar time:

```

    int tm_sec;        /* seconds after the minute */
    int tm_min;        /* minutes after the hour */
    int tm_hour;       /* hours since midnight */
    int tm_mday;       /* day of the month */
    int tm_mon;        /* months since January */
    int tm_year;       /* years since 1900 */
    int tm_wday;       /* days since Sunday */
    int tm_yday;       /* days since January 1 */
    int tm_isdst;      /* Daylight Saving Time flag : is positive if DST is in effect, zero if not in
                        effect, negative if information not known. */

```

Note: implementations may change field order and include additional fields.

[clock_t](#) **clock**(void); Returns elapsed processor time used by program or -1 if not available.

[time_t](#) **time**([time_t](#)* *tp*); Returns current calendar time or -1 if not available. If *tp* is non-[NULL](#), return value is also assigned to **tp*.

double difftime([time_t](#) *time2*, [time_t](#) *time1*); Returns the difference in seconds between *time2* and *time1*.

[time_t](#) **mktime**([struct tm](#)* *tp*); If necessary, adjusts fields of **tp* to fall within normal ranges. Returns the corresponding calendar time, or -1 if it cannot be represented.

char* **asctime**(const [struct tm](#)* *tp*); Returns the given time as a string of the form:
Sun Jan 3 13:08:42 1988\n\0

char* **ctime**(const [time_t](#)* *tp*); Returns string equivalent to calendar time *tp* converted to local time. Equivalent to: [asctime](#)([localtime](#)(*tp*))

[struct tm](#)* **gmtime**(const [time_t](#)* *tp*); Returns calendar time **tp* converted to Coordinated Universal Time, or [NULL](#) if not available.

[struct tm](#)* **localtime**(const [time_t](#)* *tp*); Returns calendar time **tp* converted into local time.

size_t **strftime**(char* *s*, **size_t** *smax*, const char* *fmt*, const [struct tm](#)* *tp*);

Formats **tp* into *s* according to *fmt*. Places no more than *smax* characters into *s*, and returns number of characters produced (excluding terminating NUL), or 0 if greater than *smax*.

Formatting conversions (%c) are:

A /* name of weekday */	p /* local equivalent of "AM" or "PM" */
a /* abbreviated name of weekday */	S /* second [00-61] */
B /* name of month */	U /* week number of year (Sunday as 1st day of week) [00-53] */
b /* abbreviated name of month */	W /* week number of year (Monday as 1st day of week) [00-53] */
c /* local date and time representation */	w /* weekday (Sunday as 0) [0-6] */
d /* day of month [01-31] */	X /* local time representation */
H /* hour (24-hour clock) [00-23] */	x /* local date representation */
I /* hour (12-hour clock) [01-12] */	Y /* year with century */
j /* day of year [001-366] */	y /* year without century [00-99] */
M /* minute [00-59] */	Z /* name (if any) of time zone */
m /* month [01-12] */	