Ficha de Trabalho n.º 3

Objectivos: Estruturas dinâmicas - A **Pilha**.

Os exercícios propostos nesta ficha visam criar um programa que facilite a gestão de um armazém de contentores empilhados segundo uma lógica LIFO (*Last In First Out*).

1. Defina uma estrutura de dados adequada para manter em memória RAM os seguintes dados relativos a um contentor armazenado:

a) Conteúdo (80 caracteres);
b) Origem (30 caracteres);
c) Destino (15 caracteres);
d) Cliente (80 caracteres);
e) Data de entrada (**do sistema** [ano, mes, dia]);
f) Tonelagem (número real);
g) Referência (15 caracteres).

2. Defina uma estrutura de dados adequada para manter em **memória RAM** uma pilha de contentores.

3. Escreva uma função que faça o registo dos dados relativos a um contentor que chega ao armazém. Esta função deve inserir a data do sistema.

4. Escreva uma função que permita visualizar os dados relativos aos contentores que se encontram em armazém.

5. Escreva uma função que permita inserir um contentor no topo da pilha.

6. Escreva uma função que permita retirar um contentor do topo da pilha.

7. Escreva uma função que permita saber quantos contentores tem uma pilha.

8. Escreva uma função que permita saber qual o contentor que se encontra no fundo da pilha.

9. Escreva uma função **menu()** que oriente o utilizador do programa na escolha das diferentes funcionalidades que lhe são proporcionadas.

10. Escreva um programa que integre as funções descritas nos números anteriores.

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