1. [9,5] Cada instância do tipo `szarray_t` representa um *array* genérico redimensionável manualmente. Na construção é indicado o *sizeof* do tipo dos elementos a armazenar, bem como a capacidade inicial em número de itens. Para alterar a capacidade (seja para aumentar ou diminuir), utiliza-se a operação *redim*. A macro *idx* permite aceder a uma posição do *array* de forma tipificada.

As declarações em C apresentadas ao lado serão a base para as respostas às alíneas seguintes.

szarray.h

```
#ifndef SZARRAY_H
#define SZARRAY_H

typedef struct szarray {
   unsigned capacity; /* capacidade máxima corrente */
   unsigned nelems;   /* número de elementos actual */
   unsigned esize;    /* sizeof de cada entrada     */
   void *   data;     /* espaço de armazenamento    */
} szarray_t;

/* Inicializa instância, alocando o espaço interno para
   a capacidade indicada. */
void szarray_init(szarray_t * obj, unsigned esize,
                                   unsigned capacity);

/* Liberta recursos internos. */
void szarray_clean(szarray_t * obj);

/* Modifica a capacidade máxima corrente, possivelmente
   perdendo elementos. */
void szarray_redim(szarray_t * obj, unsigned capacity);

/* Acesso tipificado a um índice do array. Por exemplo:
     double val = szarray_idx(dbl_arr, 3, double);
     char * str = szarray_idx(str_arr, 8, char *);
     szarray_idx(int_arr, 2, int) = 88;
     szarray_idx(str_arr, 7, char *) = "PSC"; */
#define szarray_idx(POBJ, IDX, TYPE) ...

/* Copia para dst os elementos de src para os quais
   predicate retorne valor diferente de zero.
   Retorna o número de elementos copiados. */
int szarray_extract(void * dst, szarray_t * src,
                    void * context,
   int (*predicate)(void * item, void * context));

#endif
```

a) [1.5] Implemente a operação `szarray_init`, que inicializa uma instância de `szarray_t`, deixando alocado espaço para a capacidade indicada.

b) [0.5] Escreva a função `szarray_clean`, que liberta os recursos internos da instância de `szarray_t`.

c) [1.5] Construa a função `szarray_redim`, que modifica a capacidade corrente, preservando-se os elementos já existentes (até à posição correspondente à nova capacidade).

d) [1.25] Defina a macro `szarray_idx` para dar acesso tipificado a uma posição do *array*, sem verificações.

e) [1.25] O quadro `fx.s` apresenta a implementação, em *assembly* IA-32, de uma função que recebe como argumento um ponteiro para `szarray_t`. Apresente uma versão C da função `fx`.

fx.s

```
    .text
    .intel_syntax noprefix

    .global fx
fx:
    mov eax, [esp + 4]
    mov eax, [eax + 4]
    ret

    .end
```

f) [3.5] Implemente, em *assembly* IA-32, a função `szarray_extract`, que copia para `dst` os elementos de `src` para os quais a função `predicate` retorne um valor diferente de zero. No final é retornado o número de elementos copiados. Os argumentos da função `predicate` são o endereço do elemento a avaliar e o ponteiro recebido em `context`.

2. Considere as tabelas que apresentam um excerto das secções e tabelas de símbolos de um executável e de dois módulos objecto.

m1.o

Secções:

Name	*Size*	*VMA*
.text	00000005	00000000
.data	00000500	00000000
.bss	00000000	00000000

Tabela de símbolos:

```
00000000 t f1
00000000 U i
00000020 D vals2
00000000 C j
```

m2.o

Secções:

Name	*Size*	*VMA*
.text	00000017	00000000
.data	00000300	00000000
.bss	00000000	00000000

Tabela de símbolos:

```
         U f1
00000000 T f2
00000000 d i
0000000d T main
00000100 D vals
000002FC D vals2
```

Executável

Secções:

Name	*Size*	*VMA*
.text	00001000	08048300
.data	00000600	08049300
.bss	00000008	08049900

Tabela de símbolos:

```
080483cc T f1
080483b4 T f2
080483c1 T main
08049300 d i
08049304 D i
08049900 D vals
08049950 D vals2
```

```
C–common (data ou bss)
B–bss D–data
T–text U–undefined
Maiúscula-global
Minúscula-local
VMA – Endereço Inicial da secção
Size – Dimensão da secção
```

Legenda

a) [1,5] Para cada símbolo correspondente do módulo `m1` indique uma definição em C que o possa originar.

b) [1,5] Indique se o executável apresentado pode ter resultado da ligação estática de `m1.o` e `m2.o`. Justifique.

3. [1.5] Uma biblioteca pode ser disponibilizada em forma binária para ligação estática (ficheiro `.a`) ou dinâmica (ficheiro `.so`) com os programas que dela precisem. Explique de forma sucinta, mas clara, os aspectos relacionados com a ocupação de espaço em disco associados a cada uma das opções.

4. [1.5] Existe uma situação em que um contexto obtido com `setjmp` fica inválido para ser utilizado por `longjmp`, mesmo sem ter sido modificado. Que situação é esta e qual a razão que o torna inválido?

5. [4.5] Considere o código em C apresentado a seguir. Defina os tipos, variáveis e funções correspondentes à implementação do tipo `LessThanIntFilter`, que implementa a interface `Filter` indicada, por forma a que a função eval retorna um valor diferente de zero quando o item avaliado tiver um valor inferior ao indicado no construtor. A execução do programa de teste deverá apresentar no ecrã os valores: **`6 -11 0`**. Note que <u>não</u> se considerará razoável que a implementação de `LessThanIntFilter` mantenha estado fora das respectivas instâncias.

```
#include <stdio.h>

typedef struct filter Filter;

typedef struct filter_methods {
   void (*clean)(Filter *);
   int  (*eval) (Filter *, void *);
} FilterMethods;

struct filter {
   FilterMethods * vptr;
};

void delete_Filter(Filter * filter) {
   if (filter) {
      filter->vptr->clean(filter);
      free(filter);
   }
}

/* AS DEFINIÇÕES DE LessThanIntFilter FICARÃO AQUI */

void print_filtered(int * items, unsigned len, Filter * filter) {
   unsigned idx;
   for (idx = 0; idx < len; ++idx) {
      if (filter->vptr->eval(filter, &items[idx]) != 0) {
         printf("%d ", items[idx]);
      }
   }
   putchar('\n');
}

int data[] = { 8, 6, -11, 9, 0 };

int main() {
   Filter * less_than_8 = (Filter *) new_LessThanIntFilter(8);
   print_filtered(data, 5, less_than_8);
   delete_Filter(less_than_8);
   return 0;
}
```

Duração: 2 horas e 30 minutos

Bom teste!