1. [3,5] Considere os ficheiros fonte:

a.c
```
#include <dlfcn.h>
#include <stdio.h>
#include "xy.h"
typedef int (*Fx)();
int y=40, z=50;
int main()
{ void *h,*g; Fx r,s; int i,j,k;
  h=dlo("./b.so");g=dlo("./c.so");
  r=dlsym(h,"f");s=dlsym(g,"f");
  i=r(); j=s(); k=r();
  printf("%d %d %d\n",i,j,k);
  dlclose(g);dlclose(h);
  return 0; }
```

b.c
```
#include "xy.h"
int y=20;
static int z;
int f()
{ return ++x + fx() + ++z; }
```

c.c
```
#include "xy.h"
int y=30;
static int z;
int f()
{ return ++x + fx() + ++z; }
```

xy.h
```
static int x = 10;
extern int y;
static int fx() {return ++y;}
#define dlo(S) dlopen(S,RTLD_NOW)
```

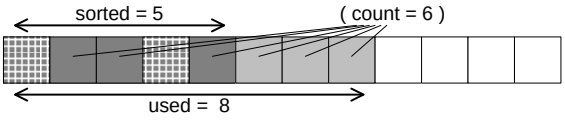
Makefile
```
a: a.c xy.h b.so c.so
  gcc -Wall -o a a.c -ldl
b.so: b.c xy.h
  gcc -fPIC -shared -o b.so b.c
c.so: c.c xy.h
  gcc -fPIC -shared -o c.so c.c
```

a) [1] Da execução de `make` resultam *warnings*, em `a.c`, sobre dois elementos de `xy.h`. Quais e porquê?

b) [1] Que símbolos públicos e definidos estão presentes em `a.o` e `b.o`, por compilação de `a.c` e `b.c`?

c) [1,5] Considerando o único executável produzido, indique e justifique o *output* da sua execução..

2. [10] Para implementar uma lista negra de cartões, definiram-se os ficheiros `BadCards.h` e `BadCards.c`. Para acelerar as pesquisas, o *array* de cartões (campo `slots` de `BC`) está ordenado até ao índice `sorted-1`. Para acelerar as inserções, estas são feitas no fim de forma não ordenada. Assim, as pesquisas, realizadas em `bc_find`, usam a função de pesquisa binária da biblioteca C (`bsearch`) apenas na parte ordenada. Ocasionalmente, deve ser chamada a função `bc_compact`, para ordenar os dados.



BadCards.h
```
#define MAX_SLOTS 4096
#define MAX_CARDLEN 16

#define VALID   'A'
#define INVALID 'Z'

typedef struct card {
  char state;       /* VALID ou INVALID */
  char num[MAX_CARDLEN]; /* ASCII string */
} Card;

typedef struct bad_cards {
  unsigned used;   /* used slots */
  unsigned count;  /* valid slots */
  unsigned sorted; /* sorted slots */
  Card *   slots[MAX_SLOTS];
} BC;

/* retorna: 1 → sucesso; 0 → sem espaço */
int bc_add(BC * bc, const char * num);

/* retorna: -1 → não existe ; >=0 → índice */
int bc_find(BC * bc, const char * num);

/* idx em sorted ? invalidar : remover */
void bc_rem(BC * bc, unsigned idx);

/* garantir que sorted == count == used */
void bc_compact(BC * bc);
```

BadCards.cpp
```
#include "BadCards.h"
#include <stdio.h>
#include <string.h>

typedef int (*cmp_t)(const void * p1, const void * p2);

static int sort_cmp(char * * pcnum1, char * * pcnum2)
{ return strcmp(*pcnum1, *pcnum2); }

static int search_cmp(char * pnum, Card * * pcard)
{ return strcmp(pnum, (*pcard)->num); }

static void bc_clean(BC * bc) {
  unsigned i;
  for (i = bc->sorted; i < bc->used; ++i)
    free(bc->slots[i]);
  bc->used = bc->sorted;
}

void bc_compact(BC * bc) {
  qsort(bc->slots,bc->used,sizeof(Card*),(cmp_t)sort_cmp);
  bc->sorted = bc->count; bc_clean(bc);
}

int bc_find(BC * bc, const char * cnum) {
  Card **p; size_t i;
  p = (Card**)bsearch(cnum, bc->slots, bc->sorted,
                      sizeof(Card*), (cmp_t)search_cmp);
  if (p && (*p)->state == VALID) return p - bc->slots;
  for (i = bc->sorted; i < bc->used; ++i)
    { if (strcmp(cnum, bc->slots[i]->num) == 0) return i; }
  return -1;
}
```

(continua)

a) [2] Implemente em C a função `bc_add` que cria dinamicamente um instância de `Card`, copiando a *string* apontada pelo argumento `num` para o campo de `Card` com o mesmo nome e adicionando a nova instância no índice `bc->used` de `bc->slots`. Procure alocar o espaço estritamente necessário para a instância de `Card`.

b) [2] Implemente em C a função `bc_rem` que elimina a entrada presente no índice `idx` do *array* `bc->slots`. Se a entrada estiver na parte ordenada do *array*, é apenas invalidada, afectando-se o seu campo `state`; se estiver na zona não-ordenada, a entrada é eliminada, sendo reduzido o valor de `used`.

c) [1,5] Como é que a função `bc_compact` recupera o espaço ocupado pelas entradas invalidadas? Justifique detalhadamente a sua resposta.

d) [2] Apresente uma versão da função `search_cmp` em *assembly* IA-32, sem recorrer a funções auxiliares.

e) [2,5] Apresente uma versão da função `bc_clean` em *assembly* IA-32.

3. [1,5] Numa máquina com representação de inteiros a 32 *bits*, constatou-se que a seguinte variável ocupava exactamente um bloco da *cache* de dados de 128KB, 4-*way set-associative*. Nesta máquina, quantos *bits* de endereço são usados para indexar os *sets*? Justifique detalhadamente a sua resposta.

```
struct { unsigned len; int elems[15]; }
```

4. [5] Para o código apresentado a seguir, pretende-se que a execução da função `main()` resulte na escrita do que está em comentário em cada linha que chama a função `foreach()`. Para cumprir este objectivo, defina o que for necessário, incluindo o tipo `ForEach`, e reescreva a função `main()`, substituindo apenas as reticências.

```
int foreach(int *v, unsigned len, ForEach *p) {
  unsigned i;
  p->v->begin(p);
  if (len>0) {
    p->v->each(p,v[0]);
    for(i=1; i<len ; ++i) { p->v->sep(p); p->v->each(p,v[i]); }
  }
  p->v->end(p);
  return p->res;
}

int main() {
  int a[] = {1,2,3,5,7,11};
  ForEach *prt1, *prt2, *sum;
  ...
  foreach(a,6,prt1); putchar('\n');      /* [1;2;3;5;7;11] */
  foreach(a,6,prt2); putchar('\n');      /* 1 2 3 5 7 11   */
  printf("sum=%d\n",foreach(a,6,sum));   /* sum=28         */
  ...
  return 0;
}
```

Duração: 2 horas e 30 minutos

Bom teste!

Funções da biblioteca C usadas na resolução do grupo 2:

```
void *bsearch(const void *k, const void *base, size_t n, size_t size, int (*cmp)(const void *, const void *));
void qsort(void *base, size_t n, size_t size, int (*cmp)(const void *, const void *));
```

`k` – chave; `base` – endereço base do *array*; `n` – número de elementos; `size` – dimensão de cada elemento; `cmp` – função de comparação que retorna 0, positivo ou negativo se a comparação der igual maior ou menor, respectivamente.
A função `bsearch` invoca a função de comparação com a chave e um ponteiro para um elemento. Retorna o ponteiro para o elemento encontrado ou `NULL`, caso contrário. A função `qsort` invoca a função de comparação com ponteiros para dois elementos.