[1,5] Considere o seguinte programa em C:

a) [1] Defina a *macro* `G_XCHG` que troca dois elementos com tipo `T`, com índices `I1` e `I2` num *array* `P`. No exemplo, a invocação da *macro* troca o segundo com o terceiro *short*.

b) [0,5] Indique o *output* do programa.

p1.c
```
#define G_XCHG(T,P, I1, I2) ...

int main() {
  int v1[] = { 0x1234, 0x12345678, 0x12 };

  G_XCHG(short, v1, 1, v1, 2);
  printf("0x%X 0x%X 0x%X\n",v1[0],v1[1],v1[2]);

  return 0;
}
```

1. [1,5] Considere o ficheiro fonte **`p2b.c`** e o excerto da tabela de símbolos do ficheiro objecto relocalizável **`p2a.o`**.

a) [1] Para cada símbolo correspondente a variáveis no módulo **`p2a.o`**, indique uma definição em C que o possa originar.

b) [0,5] Indique os erros que ocorrem na ligação do módulo **`p2a.o`** com o módulo **`p2b.o`** e proponha uma possível correcção dos mesmos.

```
> nm p2a.o
00000000 b a
00000002 b b
00000003 b c
00000004 C d
00000000 D e
00000004 d f
00000000 T g
0000004c t h
         U i

B-bss     D-data
T-text    U-undefined
C-common
```

p2b.c
```
long f=3;

void e() {/*...*/}
void g();

static void i() {
   /*...*/
}
```

2. [2] Considere as seguintes definições:

Sabe-se que, num determinado caso, um percurso sequencial de leitura dos membros **`key`** de todos os índices de **`entries`** deixou ocupadas 1024 linhas distintas de uma cache de dados *2-way set associative*, o que corresponde a uma ocupação de 25% do espaço total disponível. Um segundo percurso sequencial, com leituras dos membros **`readings[0]`** e **`readings[1]`** de todos os índices de **`entries`**, deixou ocupadas mais 1024 linhas. Responda às seguintes questões, justificando devidamente as suas respostas:

```
struct RegEntry {
   unsigned key;
   char code[6];
   short config;
   int readings[12];
   char * notes;
};

struct RegEntry entries[1024];
```

a) [1] Quantos bits de endereço são usados para determinar o *set* da cache e quantos são usados como *offset*?

b) [1] Estando ainda disponível 50% do espaço total da cache, quantas entradas adicionais poderia ter o *array* **entries** para que o espaço fosse totalmente ocupado?

3. [6] Considerando o código apresentado no quadros abaixo, implemente o estritamente necessário relativamente ao tipos `Enumerator`, `ListEnumerator` e `ArrayEnumerator` para que a execução do programa produza o *output* apresentado :

```
#include <stdio.h>
typedef struct node { struct node * next; int value; } Node;
Node nt = { NULL, 555 }, nm = { &nt, -39 }, nh = { &nm, 181 }, * list = &nh;
ListEnumerator le;

int data[] = { 912, 231, -17, 141, -11 };
ArrayEnumerator ae;

Enumerator * enumerators[] = { (Enumerator *)&le, (Enumerator *)&ae };

void showAllDecHex(Enumerator * * enums, unsigned n) {
    for ( ; n > 0; --n, ++enums) {
      while ((*enums)->vptr->moveNext(*enums))
         printf("[%d | 0x%x]\n", (*enums)->vptr->getCurrent(*enums),
                                 (*enums)->vptr->getCurrent(*enums));
      putchar('\n');
    }
}
```

output
```
[181 | 0xb5]
[-39 | 0xffffffd9]
[555 | 0x22b]

[912 | 0x390]
[231 | 0xe7]
[-17 | 0xffffffef]
[141 | 0x8d]
[-11 | 0xfffffff5]
```

```
int main() {
    initListEnumerator(&le, list);
    initArrayEnumerator(&ae, data, 5);
    showAllDecHex(enumerators, 2);
    return 0;
}
```

4. [2] Considere um cenário em que um programa tem a possibilidade de ser estendido por via de *plugins*, implementados em bibliotecas de ligação dinâmica, havendo, no entanto operações comuns (código), que são necessárias quer no módulo executável central, quer nas bibliotecas de ligação dinâmica. Indique duas soluções distintas para evitar ter réplicas do mesmo código nos vários módulos, explicando claramente de que forma cada uma das soluções resolve o problema.

5. [7] Na realização de um programa para gerir expressões aritméticas sobre inteiros, considere as seguintes definições em C:

exp.h

```
typedef struct exp {
   int (*exec)(struct exp *this);
   const char * desc;
   int * argv;
   char argc;
} Exp;

typedef struct res {
   int r; /* res. da exp */
   /* índice da expressão  que
      gerou o res.
   */
   int exp_idx;
} Res;

int exec1_exp(Exp *pe);
```

exec.s

```
.intel_syntax noprefix
.global exec1_exp

  mov    eax,[esp+0x4]
  mov    eax,[eax+0x8]
  mov    edx,[eax]
  mov    eax,[eax]
  sar    edx,0x1f
  xor    eax,edx
  sub    eax,edx
  ret
```

exp.c

```
#define MAX_EXPRESSIONS 100
/* Colecção de expressões */
static Exp * exps[MAX_EXPRESSIONS];
static int exps_sz = 0;

/* Afecta o argumento argi da expressão i com o valor argv */
int setArg(int i, int argi, int argv) { … }

/* Executa a expressão no índice i, retornando em out o resultado da expressão.
   A função retorna true em caso de sucesso da operação */
int exec(int i, int * out) {
   if (i >= 0 && i < exps_sz) {
      *out = exps[i]->exec(exps[i]); return 1;
   }
   return 0;
}

/* Cria e constrói uma nova expressão adicionando-a à colecção de expressões */
void exp_new(int (*exec)(Exp *this), int *argv, char argc, const char *desc)
{…}

/* Remove e elimina a última expressão da colecção de expressões */
void remove_last() { free((void*)exps[--exps_sz]->desc); free(exps[exps_sz]); }

/* Executa as expressões da colecção de expressões de acordo com um determinado
critério. Retorna um array com os resultados */
Res * exec_by_class(int expFilter(Exp* exp,void * filterCtx),
                    void *filterCtx,
                    int * res_size) {…}
```

Uma expressão `Exp` é constituída por uma função que a sabe calcular (**`exec`**), por uma descrição (**`desc`**), por um conjunto variável de argumentos (**`argv`**) e pelo número total de argumentos (**`argc`**).

a) [1] Implemente em C a função `exec1_exp` que já está implementada em IA-32 no ficheiro `exec.s`.

b) [2] Implemente em C função `exp_new` que cria e adiciona uma nova expressão, devidamente iniciada, à colecção de expressões `exps`.

c) [2] Implemente em *assembly* IA-32 a função `exec`.

d) [2] Realize em C a função

`Res* exec_by_class(int expFilter(Exp* exp,void * filterCtx), void *filterCtx, int *res_size)`

que executa apenas as expressões da colecção de expressões `exps` que satisfaçam um determinado critério (filtro). O parâmetro `expFilter` define o filtro a aplicar. O parâmetro `filterCtx` define o contexto adicional para o filtro. O filtro retorna 1 caso a expressão obedeça ao critério e 0 em caso contrário. A função retorna um *array* de elementos do tipo `Res`, onde cada elemento indica o índice da expressão avaliada e o respectivo resultado. O parâmetro de saída `res_size` indica a dimensão do *array* retornado.

Duração: 2 horas e 30 minutos

Bom teste!