Teste Global, 11 de Fevereiro de 2010

```
int main() {
  char *t = "ISEL";
  int a[]= {20,10,5,7,9};
  PRINT_ARRAY(t,"%c",4);
  PRINT_ARRAY(a,"%d,",5);
  return 0;
} /* output: ISEL20,10,5,7,9 */
```

1. [2] Considere o seguinte função `main()`:

a) [1] Defina a macro `PRINT_ARRAY` para apresentar no standard output os elementos do *array* (de tipos primitivos) indicado como primeiro parâmetro segundo a formatação especificada no segundo parâmetro. O terceiro parâmetro indica a dimensão do *array*.

b) [1] Defina uma função `PRINT_ARRAY` com os mesmos parâmetros e o mesmo objectivo, ou caso entenda que não é possível, justifique.

2. [10] Na realização de um programa para gerir as estadias de um hotel, considere os ficheiro `Hotel.h` e `Hotel.c`. Cada estadia armazena o nome do cliente e os números do quarto e do andar. A variável global `stays` é um *array* bidimensional de ponteiros para estadias que estão a `NULL` se o respectivo quarto está livre. As estadias são alojadas dinamicamente por cada entrada de cliente no hotel.

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>

#define ROOMS ... /* alínea a) */
#define FLOORS  4

typedef struct _stay {
  char *name;    /* Nome do cliente */
  int floor;    /* num. do andar 0..N */
  int room;    /* num. do quarto 0..N */
} Stay; /* Estadia */

extern Stay * stays[FLOORS][ROOMS];

void enter(char *name, int floor, int room);
void leave(int floor, int room);
Stay * stay(int floor, int room);
Stay * findStay(char *name);
Stay * procStays(int (*fx)(Stay *));
void setStay(Stay *s, char *n, int f, int r);
void leaveAll();
```

```
#include "Hotel.h"

Stay * stays[FLOORS][ROOMS];

Stay * stay(int f, int r) { return stays[f][r]; }

Stay *procStays(int (*fx)(Stay *)) {
  Stay *s, **ss= (Stay**)stays;
  for( ; ss < ((Stay**)stays)+FLOORS*ROOMS ; ++ss)
    if ((s=*ss) && fx(s)) return s;
  return 0;
}

static int freeStay(Stay *s)
{free(s->name); free(s); return 0;}
void leaveAll() { procStays(freeStay); }

void enter(char *name, int floor, int room) { ... }
void leave(int floor, int room)
{ freeStay(stays[floor][room]); stays[floor][room]=0; }

int main() { ... }
```

a) [1] Sabendo que o código em *assembly* da função `stay` é o indicado. Qual é o valor da constante `ROOMS`?

```
stay:
  push  ebp
  mov   ebp, esp
  mov   eax, [ebp+8]
  lea   eax, [eax+eax*4]
  add   eax, eax
  add   eax, [ebp+12]
  pop   ebp
  mov   eax, [stays+eax*4]
  ret
```

b) [1] Devido a um *bug*, o programa acede ao *array* `stays` com a expressão `stays[2][-2]`. Qual o elemento do *array* que é realmente acedido?

c) [2] Apresente uma implementação em IA-32 da função `procStays`.

d) [2] Descreva todo o conteúdo do ficheiro `findStay.c` com a implementação da função `findStay` usando a função `procStays`. A função `findStay` retorna um ponteiro para a primeira estadia encontrada do cliente com o nome indicado, ou retorna NULL se não encontrou nenhuma com esse cliente.

e) [2] Implemente em IA-32 a função `setStay` num módulo `setStay.s`. Esta função preenche uma estrutura `Stay` com os elementos indicados (n-`name`, f-`floor` e r-`room`) copiando o nome do cliente para memória alojada dinamicamente através da função `char * strdup(const char*s)`. A função `strdup` retorna um ponteiro para o espaço alojado dinamicamente contendo uma cópia da *string* passada como parâmetro.

f) [1] Implemente em C a função `enter` que pertence ao módulo `Hotel.c`. Esta função usa a função `setStay` e acrescenta uma estadia no *array* `stays` que será mais tarde removida com a função `leave`. Se a estadia para o andar e quarto indicados já existir, ela será apenas actualizada.

g) [1] Faça o *makefile* para gerar o programa completo, compilando separadamente cada um dos módulos.

3. [8] Pretende-se que implemente o módulo `tree.o`. De acordo com o `tree.h` apresentado a seguir, um `BNode` representa um nó da árvore binária e é constituído por um ponteiro para ponteiro para `BNode` (`parent`), um ponteiro para a sub-árvore esquerda (`left`), um ponteiro para a sub-árvore direita (`right`) e por um ponteiro para um elemento genérico (`pdata`). O ponteiro `parent` do nó filho referencia o campo ponteiro do pai que referencia o próprio filho. De acordo com o `list.h` apresentado a seguir, um `LNode` representa um nó de uma lista simplesmente ligada e é constituído por um ponteiro para o próximo nó na lista (`next`) e por um ponteiro para um elemento genérico (`pdata`).

```c
#ifndef _tree_h
#define _tree_h
#include "list.h"
typedef struct bnode_t {
        struct bnode_t ** parent;
        struct bnode_t * left;
        struct bnode_t * right;
        void * pdata;
} BNode;
int xpto(BNode *t);
void free_tree(BNode *t);
void remove_le(BNode *t, void *e, int (*cmp)(void*,void*));
LNode * remove_above_height(BNode *t, int h);

#endif/*_tree_h*/
```

```c
#ifndef _list_h
#define _list_h
typedef struct lnode_t {
        struct lnode_t * next;
        void * pdata;
} LNode;

#endif/*_list_h*/
```

a) [1] Analisando o código da função recursiva `int xpto(BNode *p)`, descreva de forma não recursiva, justificando sucintamente, o resultado obtido com a chamada a esta função.

```c
int xpto(BNode *t) {
        int v1, v2;
        if (t == NULL) return 0;
        v1 = xpto( t->left );
        v2 = xpto( t->right );
        return ( v1 > v2 ? v1 : v2 ) + 1;
}
```

b) [1] Implemente a função `void free_tree(BNode *t)` que liberta o espaço ocupado por todos os nós da árvore *t*. O espaço ocupado pelos elementos genéricos referenciados pelos nós não é libertado.

c) [2] Implemente a função `void remove_le(BNode *t, void *e, int (*cmp)(void*,void*))` que remove da árvore *t* todos os elementos menores ou iguais ao elemento *e* de acordo com o critério de comparação especificado em `cmp`. A função especificada em `cmp` retornará um valor positivo, zero ou negativo se o primeiro argumento for superior, igual ou inferior ao segundo argumento, respectivamente.

d) [3] Implemente a função `LNode * remove_above_height(BNode *t, int h)` que remove da árvore *t* todos os elementos que estejam acima da altura especificada em *h*. A função deverá retornar o ponteiro para o primeiro nó da lista simplesmente ligada constituída pelos elementos removidos da árvore. Os elementos deverão ficar na lista com a ordem inversa com que estão na árvore. Uma árvore perfeitamente balanceada com altura 0 tem 0 elementos, com altura 1 tem 1 elemento, com altura 2 tem no máximo 3 elementos, com altura 3 tem no máximo 7 elementos, …

e) [1] Faça o *makefile* que cria a biblioteca estática `tree.a` sabendo que, quando *linkada* com algum outro ficheiro objecto realocável, produzirá um executável com a menor dimensão possível dependente das funcionalidades declaradas em `tree.h` que forem usadas.

Duração: 2 horas e 30 minutos

Bom teste!