**Instituto Superior de Engenharia de Lisboa**

# Programação II

**2021/22 – 2.º semestre letivo**

**Teste de época normal**

2022.06.29

Nota importante: Valoriza-se a escrita de código que inclua comentários esclarecedores da implementação seguida e que contenha indentação legível.

## 1. [4,5 valores]

Pretende-se fazer o registo de localizações no globo terrestre. Para tal será usado o tipo **`Coord`** para fazer o registo da latitude, longitude e da distância ao ponto (0,0).

```
typedef struct {
  double lat;                   // Latitude
  double lng;                   // Longitude
  double distance;              // Distância a (0,0)
} Coord;
```

Pretende-se usar uma estrutura do tipo **`Table`** para armazenar um conjunto de coordenadas suportada pela seguinte definição:

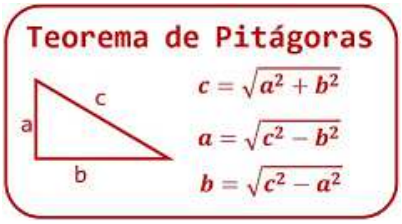
```
typedef struct {
  int count;           // N.º de elementos efetivamente na lista
  Coord arr[N_PLACES];// Array com N_PLACES elementos criados estaticamente
} Table;
```

a) [1,0] Escreva a função

```
void fill_coord( Coord *coord, double lat, double lng );
```

que preenche uma estrutura, indicada pelo parâmetro `coord`, capaz de armazenar dados relativos a uma localização indicada pelos parâmetros `lat` (latitude) e `lng` (longitude). Deve ainda ser feito o cálculo da distância ao ponto (0,0).
Sugere-se a utilização da função **`pow`** declarada em `math.h`.

```
double pow(double val1, double val2);
```




b) [1,0] Escreva a função

```
void add_coord(Table *table, Coord *coord);
```

que adiciona uma nova coordenada, com os valores indicados por `coord`, à tabela indicada por `table` (a qual se supõe já inicializada previamente com `count`=0). De notar que a quantidade máxima de elementos que a tabela pode suportar está definido na constante *N_PLACES*.

c) [1,0] Escreva a função

```
int compare_coord(const void *coord1, const void *coord2);
```

que compara duas coordenadas, retornando um valor negativo, zero ou um valor positivo, usando o seguinte critério, pela ordem a seguir indicada:

1. Distâncias ao ponto (0,0) são comparadas de forma crescente;
2. Distâncias iguais, desempatam com valores de latitude de forma crescente;
3. Distâncias e latitudes iguais, desempatam com valores de longitude de forma crescente.

d) [1,5] Escreva a função

```
Coord * search_coord(Table *table, double lat, double lng);
```

que verifica se a coordenada (`lat`, `lng`) existe na tabela (`table`) de coordenadas. Para tal, deverá:

- Ordenar a lista recorrendo à função **`qsort`** da biblioteca *standard* da linguagem;
- Realizar uma pesquisa binária ao *array* já ordenado, usando para tal a função **`bsearch`** da biblioteca *standard* da linguagem.

A função deverá devolver o endereço do elemento na lista ou NULL caso não exista.

## 2. [4,5 valores]

Considere as seguintes estruturas, especificadas para a realização das Séries de Exercícios 3 e 4, representadas na caixa seguinte. Recorde o parágrafo do enunciado de uma das séries sobre uma das estruturas:

"No (...) tipo `DinRef_t` o campo `refs` é um ponteiro, destinado a apontar para um *array* de ponteiros, de modo a permitir o seu alojamento e realojamento dinâmico, para se adaptar automaticamente à quantidade de *tags* existente. O campo `space` serve para controlar a quantidade de elementos alojados, de modo a permitir o realojamento por blocos com o objetivo de evitar o custo de um realojamento por cada elemento adicionado."

Antes de iniciar a realização deste grupo, leia o enunciado de todas as alíneas que o compõem.

```
1   typedef struct{
2     char title[MAX_TIT + 1];
3     char artist[MAX_ART + 1];
4     char album[MAX_ALB + 1];
5     short year;
6     char comment[MAX_COM + 1];
7     char track;
8     char genre;
9   } MP3Tag_t;
10
11  typedef struct{
12    int space;       // Quantidade de elementos alojados no campo refs.
13    int count;       // Quantidade de elementos preenchidos no campo refs.
14    MP3Tag_t **refs; // Ponteiro para array de ponteiros para MP3Tag_t.
15                     // O array e estruturas MP3Tag_t têm aloj. dinâmico.
16  } DinRef_t;
17
```

a) [1,5] Pretende-se o desenvolvimento da função

`int count_year_range(DinRef_t *dr, short low, short high);`

que, quando executada, admite que o "*array*" de referências (passado no 1.º parâmetro) está ordenado ascendentemente pelo campo `year` das *tags* referenciadas, e retorna a contagem da ocorrência de *tags* com o membro `year` compreendido entre os parâmetros `low` e `high` (incluindo ambos). Valoriza-se a eficiência.

b) [1,5] Pretende-se o desenvolvimento da função

`MP3Tag_t *insert_ref(DinRef_t *dr, MP3Tag_t *tag);`

que, quando executada, admite que o "*array*" de referências (passado no 1.º parâmetro) está ordenado ascendentemente pelo campo `year` das *tags* referenciadas, e insere neste "*array*" o novo elemento passado no 2.º parâmetro, mantendo o "*array*" ordenado (por `year`). A função retorna, em caso de sucesso, um ponteiro para o *tag* inserido; em caso de insucesso (por exemplo, se ocorrer falta de memória ao alojar dinamicamente), a função retorna `NULL`. Sugere-se que percorra o "`array`" sequencialmente para identificar o ponto de inserção do novo elemento. Aceitam-se repetições, ou seja, antes e após a inserção poderão coexistir, no "*array*" referenciado por `dr`, mais do que uma referência para o mesmo *tag*. Não pode usar a função `qsort` da biblioteca normalizada.

c) [1,5] Pretende-se o desenvolvimento da função

```
DinRef_t *merge_refs(const DinRef_t *dr1, const DinRef_t *dr2);
```

que, quando executada, admite que os "*arrays*" de referências (passados por parâmetro) estão ordenados ascendentemente pelo campo `year` das *tags* referenciadas, e cria um novo "*array*" que contém a fusão dos "*arrays*" passados por parâmetro. O novo "*array*" deve também estar ordenado ascendentemente por `year`. A função retorna, em caso de sucesso, um ponteiro para uma estrutura `DinRef_t` que representa o novo "*array*"; em caso de insucesso (por exemplo por falta de memória dinâmica), a função retorna `NULL`. Tal como na alínea anterior, aceitam-se repetições, ou seja, poderão coexistir, no "*array*" criado e retornado pela função, mais do que uma referência para o mesmo *tag*, e não é permitida a utilização da função `qsort` da biblioteca normalizada.

## 3. [4 valores]

Pretende-se implementar conjuntos de valores numéricos, na forma de listas ligadas e respetivas funções de manipulação. Os elementos dos conjuntos são representados pelo tipo **`SetNode`**.

```
typedef struct setNode {
    struct setNode *next;  // ligação em lista
    int value;             // número representado
} SetNode;
```

As listas ligadas que representam os conjuntos são ordenadas por valor crescente dos elementos. De acordo com o conceito matemático de conjunto, não há valores repetidos. Na implementação das funções pretendidas, pode usar outras das especificadas, bem como escrever funções auxiliares. Valoriza-se a eficiência.

a) [1] Escreva a função

```
SetNode *setInsert( SetNode *s, int v );
```

que insere um elemento com o valor `v` (se não existir) no conjunto representado pela lista indicada por `s`. A função retorna o ponteiro de acesso à lista, eventualmente modificado pela inserção.

b) [1,5] Escreva a função

```
int setFind( SetNode *s, int v, SetNode **last );
```

que procura o elemento com o valor `v` no conjunto representado pela lista indicada por `s`. A função retorna 1 se o valor `v` existe ou 0 no caso oposto. O parâmetro `last`, se for NULL deve ser ignorado; caso contrário, representa o endereço de um ponteiro, o qual deve ser afetado com o endereço do último nó observado na pesquisa, pelo que este nó o contém o valor procurado ou o mais baixo dos valores superiores.

c) [1,5] Escreva a função

```
SetNode *setIntersect ( SetNode *s1, SetNode *s2 );
```

que modifica o conjunto indicado por `s1` de modo que ele passe a representar a interseção dos dois conjuntos originais, indicados por `s1` e `s2`. Retorna o acesso ao conjunto modificado. Não modifica o conjunto indicado por `s2`.

## 4. [3 valores]

Considere uma árvore binária de pesquisa com dados genéricos, formada por nós com o tipo **`TNode`** seguinte.

```
typedef struct tNode{
    struct tNode *left, *right;    // ponteiros de ligação na árvore
    void *data;                    // ponteiro para o bloco de dados
} TNode;
```

Pretende-se dispor de funções para identificar aspetos da topologia das árvores existentes, como por exemplo, entre outros, a altura ou o estado de balanceamento.

a) [1] Escreva a função

```
int tHeight( TNode *r );
```

que retorna a altura da árvore identificada pela raiz `r`. Entende-se por altura da árvore o número de ligações desde o ponteiro raiz até ao nó-folha mais distante. Por exemplo, a altura de uma árvore é: 0, se está vazia; 1, se contém um elemento apenas; 2 se contém três elementos, balanceados.

b) [1] Uma árvore considera-se desbalanceada se algum dos seus elementos tem subárvores com alturas diferentes, sendo essa diferença superior a 1. Considere a função seguinte, com uma implementação rudimentar da verificação de balanceamento,

```
int tUnbalanced( TNode *r ){
    if( r == NULL )
       return 0;
    int h1 = tHeight( r->left );
    int h2 = tHeight( r->right );
    int dif = abs( h1 - h2 );
    if( dif > 1 || tUnbalanced( r->left ) || tUnbalanced( r->right ) )
       return 1
    return 0;
}
```

Admita que a função `tUnbalanced` é chamada recebendo em `r` a raiz de uma árvore com 7 nós, perfeitamente balanceada. Tendo em conta os algoritmos recursivos, indique: (1) a quantidade total de vezes que a função `tUnbalanced` é chamada, incluindo a primeira; (2) a quantidade total de vezes que a função `tHeight` é chamada; (3) a soma destas duas quantidades.

c) [1] Tendo como objetivo executar menos chamadas recursivas, escreva uma nova versão da função

```
int tUnbalanced2 ( TNode *r, int *height );
```

que retorna 0 se a árvore identificada pela raiz `r` está balanceada ou 1 se está desbalanceada. Esta função identifica adicionalmente a altura da árvore, depositando o seu valor na variável apontada pelo parâmetro `height`. O propósito é utilizar a altura calculada pela própria função, de modo a evitar as numerosas chamadas a `tHeight`.

## 5. [4 valores]

Considere um conjunto de módulos escritos em linguagem C, **comp1.c**, **comp2.c** e **arrint.c**, compilados individualmente com o comando "**gcc -c *.c**".
As caixas abaixo apresentam, respetivamente, as definição relativas ao tipo **ArrInt** e o resultado do comando "**nm *.o**" que mostra, a partir dos módulos compilados, as listas de símbolos, classificados com as abreviaturas: **T**, *public text*; **t**, *private text*; **U**, *undefined*.

```
#define MAX_SIZE 100

typedef struct{
    int cnt;
    int arr[MAX_SIZE];
    int (*cmp)( const void *, const void * );
} ArrInt;
```

```
arrint.o:
0000000000000000 T aInit
000000000000004b T aInsert
000000000000008f T aPrint
                 U printf
                 U qsort

comp1.o:
0000000000000000 T comp_f1

comp2.o:
0000000000000000 T comp_f2
```

Considere também o módulo de aplicação, **aplic.c**, com as funções seguintes.

```
void demo(){
     ArrInt data;
     int x;
     aInit( &data, comp_f1 );
     while( scanf( "%d", &x ) == 1 ){
         if( aInsert( &data, x ) == 0 )
             printf( "Storage full\n" );
     }
     aPrint( &data );
}

int main(){
     demo();
     return 0;
}
```

Admita que são criados e usados no processo de compilação os *header files* **comp1.h**, **comp2.h** e **arrint.h**, cada um contendo as definições de tipo e assinaturas das funções públicas do módulo fonte (**.c**) com o respetivo prefixo.

a) [1] Indique, justificando, se o atributo `static`: (1) foi ou não usado nos módulos fonte **comp1.c**, **comp2.c** e **arrint.c**; (2) pode ou não ser usado no módulo de aplicação.

b) [1] Tendo em conta a sua resposta à alínea a), e admitindo que compila o módulo de aplicação com o comando "**gcc -c aplic.c**", apresente a lista de símbolos produzida por "**nm aplic.o**" e a respetiva classificação (**t**, **T** ou **U**). Por cada um dos símbolos *undefined*, indique qual o módulo que o resolve ou se é resolvido pela biblioteca.

c) [1] Escreva o *header file* **arrint.h**, tendo em conta o controlo de inclusão múltipla e considerando assinaturas das funções compatíveis com a utilização no módulo de aplicação. Indique, justificando, quais os módulos de código fonte (**.c**) onde o *header file* **arrint.h** deve ser incluído.

d) [1] Com base nos símbolos listados e nos que identificou, indique os módulos que devem ser ligados ao módulo de aplicação **aplic.o** para gerar o executável. Escreva um *makefile* que produza, de forma eficiente, o executável com o nome "**aplic**" a partir dos módulos fonte (**.c**) necessários.