**Instituto Superior de Engenharia de Lisboa**

# Programação II

**2022/23 – 1.º semestre letivo**

**Teste de época de recurso**

2023.01.12

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

## 1. [4 valores]

Considere as definições das funções `f1`, `f2`, e `f3`. Na linha 1 do troço de código seguinte deverá inserir o algarismo das unidades do seu número de aluno no local indicado. Por exemplo: se o seu número de aluno fosse o 12345, aquela linha ficaria assim: `#define ALG 5`

Nota importante: a utilização do algarismo errado penalizará significativamente a avaliação da alínea b) deste grupo.

```
1  #define ALG algarismo_das_unidades_do_seu_número_de_aluno
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  int f2(int * e) {
7     return *e == ALG;
8  }
9
10 void f3(int * e) {
11    *e *= 2;
12 }
13
14 int * f1(int a[], size_t *size,
15           int (*cond)(int * e), void (*proc)(int * e)){
16    size_t i = *size;
17    while(i--)
18       if( cond(&a[i]) ){
19          if((a=realloc(a, (*size + 1) * sizeof(int))) == NULL)
20             return NULL;
21          memmove(&a[(*size)++], &a[i], sizeof(int));
22          proc(&a[i]);
23       }
24    return a;
25 }
```

Na caixa seguinte consta um extrato do *output* do comando `man memmove` executado em consola.

```
(…)
  void *memmove(void *dest, const void *src, size_t n);


DESCRIPTION
  The memmove() function copies n bytes from memory area src to memory area
dest. The memory areas may overlap: copying takes place as though the bytes
in src are first copied into a temporary array that does not overlap src or
dest, and the bytes are then copied from the temporary array to dest.
(…)
```

No troço de programa seguinte consta um exemplo de utilização das funções `f1`, `f2`, e `f3`.

```
...   int *a;
...   (…) // Alojamento e preenchimento do array dinâmico a.
30    size_t i, size=22;
31    int *b = f1(a, &size, f2, f3);
32    for(i=0; i<size; i++)
33       printf("%d ", b[i]);
```

a) [1] Comente o código da função `f1` e apresente, correta e completamente preenchido, o respetivo cabeçalho descritivo seguindo o formato do que se apresenta na caixa seguinte, que está preenchido (como exemplo) para a função `memmove`. **Atenção**: no campo "`Descrição`" não se pretende que explique o código da função `f1` mas sim que descreva a funcionalidade (ou a utilidade) da função `f1`, independentemente do uso que lhe é dado num caso concreto.

```
/*----------------------------------------------------------------------
Nome da função: memmove

Descrição:
Esta função copia n bytes da memória apontada por src para a memória
apontada por dest. As áreas de memória podem sobrepor-se: a cópia ocorre
como se os bytes a copiar fossem primeiro copiados para um array
temporário, não sobreposto às áreas de memória apontadas por src e dst, e
fossem depois copiados para a memória apontada por dest.

Parâmetros:
void *dest: endereço da memória para onde é realizada a cópia.
const void *src: endereço da memória de onde é realizada a cópia.
size_t n: número de bytes a copiar.

Retorno:
void *: retorna o endereço apontado por dest.
----------------------------------------------------------------------*/
```

b) [1] Apresente, justificando, o *standard output* produzido pela execução do troço de código anterior, admitindo que (imediatamente antes da execução da linha 31) o conteúdo do *array* **a** é o seguinte:

| 2 | 1 | 3 | 0 | 4 | 7 | 8 | 6 | 7 | 1 | 8 | 9 | 5 | -9 | 5 | 3 | 0 | 6 | 2 | -1 | 9 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Nota: a ausência de justificação dos valores produzidos em *standard output* penalizará significativamente a avaliação desta alínea.

c) [2] Considere que se pretende agora realizar uma função `f1a` com a mesma funcionalidade da função `f1` mas com maior generalidade, que possa operar sobre *arrays* de outros tipos e não apenas *arrays* de elementos do tipo `int`. Sendo uma versão da função `f1`, a função `f1a` deve seguir **de muito perto** o algoritmo da função `f1`.

Escreva a definição da função `f1a`, adicionando ou removendo parâmetros se necessário. Escreva a definição da função com uma indentação correta, usando obrigatoriamente comentários para que fique clara a sua implementação/funcionalidade.

Rescreva a linha onde é usada a função `f1`, no exemplo de utilização anterior, substituindo-a pela utilização da função `f1a`, igualmente aplicada ao *array* `a`.

Nota: desaconselha-se veementemente a escrita da definição desta função sem a utilização de comentários porque penalizará significativamente a avaliação desta alínea.

## 2. [6 valores]

Considere a seguinte definição de um nó de uma árvore binária de pesquisa, que representa as ocorrências de uma palavra num ficheiro. A árvore tem uma ordenação pela palavra, alfabeticamente crescente da esquerda para a direita.

```
typedef struct tNode{
   struct tNode *left, *right; // ponteiros de ligação na árvore
   char *word;                 // string, alojada dinamicamente
   int freq;                   // quantidade de ocorrências da palavra word
} TNode;
```

a) [1] Escreva a função

```
void tPrintByFreq( Tnode* r, int freq );
```

que imprime todas as palavras cuja frequência seja maior do que `freq`. As palavras devem ser apresentadas por ordem alfabeticamente crescente.

b) [2] Escreva a função

```
int tWordsTotal( Tnode* r );
```

que retorna a soma das ocorrências de todas as palavras existentes na árvore.

c) [3] Escreva a função

```
int tAddWordIf( TNode **rp, char *word, int (*pred)( TNode * ) );
```

destinada a adicionar, à árvore cujo ponteiro raiz é indicado por `rp`, uma ocorrência da palavra indicada por `word`. Se a palavra não existir, deve ser adicionada. No caso de a palavra já existir, deve ser avaliado o predicado implementado pela função `pred`, passando o nó da palavra como parâmetro; se retornar um valor com avaliação *true*, a frequência deve ser incrementada, caso contrário não há modificação da árvore. A função `tAddWordIf` deve retornar 1 se adicionou a ocorrência ou 0 no caso contrário.

## 3. [6 valores]

Pretende-se o desenvolvimento de uma *hash-table*, com resolução de colisões por lista ligada, para representar informação sobre os elementos de uma empresa. Considerando um exemplo simplificado, a informação a registar é apenas o número mecanográfico, o nome e a função desempenhada. Como valor de *hash* para indexação na *hash-table* é usado o número mecanográfico.

Propõe-se as definições seguintes a estrutura de dados com os elementos a registar, os nós das listas ligadas pertencentes à *hash-table* e o descritor desta.

```
#define HASH_TABLE_SIZE 499

typedef struct{      // Descritor de um elemento
   unsigned number; //   número mecanográfico de registo na empresa
   char *name;      //   nome; string alojada dinamicamente
   char *position;  //   função desempenhada; string alojada dinamicamente
} Member;

typedef struct hlNode{  // Nó de uma lista de elementos no mesmo índice da tabela
  struct hlNode *next;  //   ponteiros de ligação na lista
  Member *data;         //   aponta elemento de informação armazenado.
} HLNode;

typedef struct {                    // Descritor de uma hash-table
  HLNode *table[HASH_TABLE_SIZE]; //   tabela de ponteiros
} HTable;
```

Admita que a *hash-table* foi previamente alojada e preenchida com os ponteiros a NULL.

a) [1,5] Escreva a função

```
HLNode *hListAdd( HLNode *head,
                  int number, char *name, char *position );
```

que adiciona a uma lista ligada, indicada por `head`, um elemento com os dados dos restantes parâmetros. A função retorna o ponteiro cabeça de lista atualizado. Assume-se que quando é chamada, o mesmo elemento não existe na lista. Pretende-se utilizar o algoritmo mais simples e rápido na inserção.

b) [2] Escreva a função

```
Member *hListFind( HLNode *head, int number );
```

que procura na lista indicada por `head` e retorna o endereço do elemento de informação com o número `number`, ou NULL, se o número procurado não existir.

c) [2,5] Escreva a função

```
Member *hFind( HTable *ht, int number );
```

que procura na *hash-table* indicada por `ht` e retorna o endereço do elemento de informação com o número number, ou NULL, se o número procurado não existir.

Assuma que os dados foram colocados na *hash-table* através de uma função de inserçao, que não faz parte deste exercício, a qual usa o número mecanográfico para indexar e recorre à função `hListAdd` para colocar na lista do índice selecionado.

## 4. [4 valores]

Considere um conjunto de módulos escritos em linguagem C, compilados individualmente com o comando "`gcc -c *.c`". Na caixa adiante apresenta-se as listas de símbolos dos módulos compilados, resultantes do comando "`nm *.o`".

A ferramenta `nm` classifica os símbolos com as abreviaturas: `T`, *public text*; `t`, *private text*; `U`, *undefined*; `D`, *public data*; `d`, *private data*; `B`, *public BSS data*; `b`, *private BSS data*.

```
comp.o:
0000000000000000 T comp1
0000000000000000 T comp2

reverse.o:
0000000000000000 T reverse
                 U swap

sortint.o:
0000000000000000 T sortInt
                 U swap

swap.o:
                 U memcpy
0000000000000000 T swap
000000000000006a T temp
```

Considere também que há um módulo de aplicação, `aplic.c`, contendo as funções seguintes.

```
void print( int arr, int cnt ){
   while( cnt-- )
      printf( "%d\t", *arr++ );
   putchar( '\n' );
}

int main(){
   int a[ARR_SIZE];
   int c = 0;
   while( c < ARR_SIZE && scanf( "%d", a + c ) == 1 )
      ++c;
   reverse( a, c, sizeof *a );
   print( a, c );
   return 0;
}
```

Admita que são criados, podendo ser usados no processo de compilação, os *header files* `comp.h`, `reverse.h`, `sortint.h` e `swap.h`, cada um contendo as assinaturas das funções públicas do módulo fonte (`.c`) com o respetivo prefixo.

a) [1] Apresente a lista de símbolos produzida por "`nm aplic.o`" e a respetiva classificação (t, T, U, etc.). Para cada símbolo indefinido, indique qual o módulo que o resolve ou se é resolvido pela biblioteca normalizada.

b) [1] Identifique, na totalidade de módulos deste exercício, se há funções a que se possa adicionar o atributo `static`. Justifique.

c) [1] Escreva o *header file* `reverse.h`, tendo em conta o controlo de inclusão múltipla e considerando, para aspetos de assinatura, a compatibilidade com a utilização na função `main`. Indique, justificando, quais os módulos de código fonte (`.c`) onde o *header file* `reverse.h` deve ser incluído.

d) [1] Com base nos símbolos listados, tendo em conta 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`) estritamente necessários.