Instituto Superior de Engenharia de Lisboa

Programação II

2021/22 – 1.º semestre letivo

Teste de época de recurso

2022.02.23

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

1. [4 valores] Considere a definição da função `f1`.

```
 1  int * f1( int * a, size_t size, size_t i ) {
 2    while(i < size - 1){
 3      a[i] = a[i + 1];
 4      i++;
 5    }
 6    return realloc(a, i * sizeof(int));
 7  }
 8
 7  main(){
 8    int * a1, * a2, size, n;
 …    …
20
21    a2 = f1( a1, size, n );
22
23  }
```

a) [1] Admita que imediatamente antes da execução da linha 21 as variáveis **`a1`**, **`size`** e **`n`** têm os seguintes conteúdos

Array **`a1:`**

17	4	9	2	36	3

`size:` 6; **`n:`** 3

Represente o *array* apontado por **`a2`** imediatamente após a execução da linha 21.
Considerando a alteração realizada ao conteúdo do *array*, explique a funcionalidade da função `f1`.

Admita agora que, noutra execução do programa, imediatamente antes da execução da linha 21 as variáveis **`a1`**, **`size`** e **`n`** têm os seguintes conteúdos:

Array **`a1:`**

16	3	36	2	4	7	6	25	13	17	9	11

`size:` 12; **`n:`** 1

Represente o *array* **`a2`** imediatamente após a execução da linha 21.

Considere a definição das funções `f2` e `f3`, notando que a função `f2` usa a função `f1` anteriormente descrita.

```
int f3(const int e) {
  if(e < 1) return 0;
  int i = 1;
  do{
    long s = i * i;
    if(e == s)
      return 1;
    i++;
  }while(i < e);
  return 0;
}

int * f2( int * a, size_t *size, int (*cond)(const int) )
{
  size_t k=0;
  while(k < *size)
    if(cond(a[k]))
      a = f1(a, (*size)--, k);
    else
      k++;
  return a;
}
```

b) [1] Admita que numa determinada aplicação a função `f3` é chamada sobre os valores do *array* **`a1`**:

Array **`a1:`**

17	4	9	2	36	3

Indique o valor de retorno da função `f3` para cada um dos seis números inteiros contidos no *array* e explique a funcionalidade desta função.

c) [1] Admita que a função `f2` é chamada com a seguinte linha de código

```
a2 = f2(a1, &size, f3);
```

e que, imediatamente antes da execução da função `f2`, as variáveis **`a1`** e **`size`** têm os seguintes conteúdos:

Array **`a1:`**

16	3	36	2	4	7	6	25	13	17	9	11

`size:` 12

Represente o *array* **`a2`** imediatamente após a execução da função, indicando e justificando o valor que fica na variável **`size`**. Neste contexto, explique a funcionalidade da função `f2`.

d) [1] Admita que se pretende realizar uma versão mais genérica da função `f1`, que mantendo a mesma funcionalidade possa operar não apenas sobre *arrays* de números inteiros mas sobre *arrays* com qualquer tipo de dados.

Sem adicionar a utilização de qualquer função da biblioteca normalizada, escreva a definição desta função e dê um exemplo da sua utilização reescrevendo a linha 21 do troço de código apresentado no início deste grupo.

2. [4 valores] Pretende-se fazer o registo de viaturas presentes no interior de uma área restrita, usando um *array* de elementos do tipo `Car`. O *array* é implementado num descritor com o tipo `CarSet` e é ordenado, alfabeticamente crescente.

```
#define CAR_PLATE_SIZE 7

typedef struct{
    char plate[CAR_PLATE_SIZE];  // Matrícula da viatura
    short present;               // 1 = está presente; 0 = está ausente
} Car;

typedef struct{
    int count;   // número de elementos preenchidos no array cars
    Car cars[MAX_CARS];
} CarSet;
```

a) [1] Escreva a função

```
void plateNormalize( char *plate );
```

que, recebendo a *string* `plate` com uma matrícula de automóvel, normaliza o seu conteúdo de modo a ficar apenas com a sequência de algarismos e letras, sendo estas colocadas em maiúsculas. Se existirem carateres de espaço, *tab* ou hífen devem ser eliminados. Por exemplo: “01-23-AB” deve ficar “0123AB”; “45 cd 67” deve ficar “45CD67”; “ EF-89-01 ”deve ficar “EF8901”;

b) [1] Escreva a função

```
void carSort( CarSet *set );
```

que ordena, alfabeticamente crescente, o *array* de elementos do tipo `Car` pertencente ao descritor `set`. Deve utilizar a função `qsort` de biblioteca e escrever a função de comparação para passar no parâmetro `compar`.

```
void* qsort( const void *base, size_t num, size_t size,
             int (*compar)(const void *, const void *) );
```

c) [1] Escreva a função

```
Car *carFindPlate( CarSet *set, char *plate );
```

que procura, no conjunto `set`, o elemento `Car` com a matrícula `plate`. Retorna o endereço do elemento encontrado ou NULL, se não existir. Tendo em conta que o *array* está ordenado, deve realizar pesquisa binária, utilizando função `bsearch` de biblioteca e reutilizando a função de comparação anterior ou definindo uma nova se necessitar que seja diferente.

```
void* bsearch( const void *key, const void *base, size_t num, size_t size,
               int (*compar)(const void *, const void *) );
```

d) [1] Escreva a função

```
int carCheckIn( CarSet *set, char *plate );
```

que, após normalizar a matrícula recebida em `plate`, regista a sua entrada na área associada ao conjunto `set`, ativando o respetivo estado presente. Se a matrícula ainda não existir, é inserida; neste caso, deve assegurar que o *array* fica ordenado. A função normalmente retorna 1; no entanto, se o espaço do *array* estiver totalmente preenchido, não adiciona e retorna 0. Deve utilizar as funções anteriores que forem adequadas para cumprir esta especificação.

3. [5 valores] Considere a seguinte definição genérica de um nó de uma lista:

```
typedef struct m_lnode{
   void *data;              //conteúdo do nó
   struct m_lnode *next;    //ligação em lista
} LNode;
```

a) [1] Admitindo a existência de dados referenciados por *arrays* de ponteiros, genéricos, escreva a função

```
LNode *listFromArray(void **arr, int length);
```

que constrói e retorna uma lista com o conteúdo do *array* `arr`, mantendo na lista a ordem dos elementos do *array* `arr`.

b) [1,5] Escreva a função

```
LNode* listFilter( LNode* list, bool (*pred)(void*));
```

que retorna uma **nova** lista com todos os elementos da lista indicada por `list` que satisfaçam o predicado `pred`. A lista original não sofre modificação.

c) [1] Escreva a função

```
void printStringsInList( LNode* list );
```

que assume os dados representados pela lista como *strings* e imprime-as na consola.

d) [1,5] Escreva a função

```
void printStringsLongerThan3( char **arr, int length )
```

que imprime na consola as *strings* do *array* `arr` que têm mais do que 3 caracteres. Deve **obrigatoriamente** utilizar as funções das alíneas a), b) e c); pode usar outras funções auxiliares que ache pertinente e deve libertar qualquer memória dinâmica auxiliar que utilize.

4. [3 valores] Pretende-se implementar uma estrutura de dados, baseada numa árvore binária de pesquisa genérica. Considere as seguintes estruturas:

```
struct m_tnode {
   void* data;
   struct m_tnode *left, *right; //ponteiros de ligação na árvore
}TNode;

struct m_tree {
   TNode * root;
   int (data_comp*)(void* a, void* b); //comparador de data
}Tree;
```

A árvore construída para esta estrutura de dados deve estar ordenada pelo campo `data` de `TNode`, fazendo uso do comparador `data_comp` da estrutura `Tree`.

Para desenvolver algoritmos recursivos aplicáveis ao tipo `Tree`, propõe-se que estes sejam implementados em funções auxiliares, as quais podem receber, em parâmetros, o acesso a um nó da árvore e, se necessário, à função de comparação.

a) [1] Escreva a função

```
int treeCount( Tree* t );
```

que retorna quantos elementos tem a estrutura de dados.

b) [1] Escreva a função

```
void treeAdd( Tree* t, void* data);
```

que adiciona um novo elemento à arvore. O parâmetro `data` indica um elemento de dados previamente alojado e preenchido, cujo conteúdo é compatível com a função de comparação associada à estrutura indicada por `t`.

c) [1] Escreva a função

```
LNode* treeToList( Tree *t );
```

que, usando nós de lista com o tipo do exercício 3, constrói e retorna uma lista ordenada com o conteúdo da árvore `root`, mantendo, na lista, a ordem relativa que os dados têm na árvore. A árvore permanece sem alteração.

5. [4 valores] Considere o conjunto de módulos com funções utilitárias, `m1.c`, `m2.c`, `m3.c`, e `m4.c` seguintes:

m1.c

```
static void fa( int x ){
   printf( "%d\n", x );
}
void f1( int a ){
   fa( a );
}
```

m2.c

```
static void fa( int x, int y ){
   printf( "%d %d\n", x, y );
}
int f2(int a, int b ){
   f1( a );
   fa( a, b );
   return a + b
}
```

m3.c

```
void f3( int a, int b ){
    f2( a, b );
}
```

m4.c

```
void f4( int a ){
    f1( a );
}
```

a) [1] Admita que compila individualmente cada um destes módulos e que utiliza a ferramenta `nm` para observar os respetivos símbolos. Sabendo que são classificados com '`U`' os símbolos indefinidos, com '`T`' os das funções públicas e com '`t`' os das privadas, indique os símbolos listados por cada um dos módulos e a respetiva classificação.

b) [1] Pretende-se dispor de um único *header file*, para incluir nos módulos de aplicação, com as assinaturas relevantes das funções implementadas nestes módulos. Escreva o *header file* completo. Indique os motivos para se usar a diretiva `#ifndef` e para se incluir o *header file* nos próprios módulos fonte.

c) [1] Admitindo que tem um módulo de aplicação, `main.c`, que chama apenas `f2` e `f4`, indique quais são os módulos necessários para gerar o executável.

d) [1] Escreva o *makefile* adequado para compilar individualmente os módulos e gerar o executável do programa de aplicação referido.