Instituto Superior de Engenharia de Lisboa

Programação II

2020/21 – 2.º semestre letivo

Teste – época especial

2021.09.16

1. [4 valores] Considere as definições das funções **`f1`** e **`f2`**. 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 de algumas alíneas deste grupo.

```
1   #define ALG algarismo_das_unidades_do_seu_número_de_aluno
2   #include <stdio.h>
3   #include <stdlib.h>
4   int f2(unsigned char n) {
5     //printf("%d \n", n); // Ignorar na alínea indicada
6     if(n < 2) return n;
7     return n + f2(n-1);
8   }
9   int f1(int ** a, size_t * sz, unsigned char n) {
10    *a = realloc(*a, (*sz + 1) * sizeof(int));
11    return (*a)[(*sz)++] = f2(n);
12  }
13  int main() {
14    int *s, *a, i, b[] = {6, 5, 6, 5, 6, 5, 6, 5, 6, 5};
15    a = malloc(sizeof(int));
16    size_t sz = 1;
17    i = f2(b[ALG]);
18    printf("i = %d\n", i);
19    i = f1(&a, &sz, b[ALG]);
20    printf("i = %d *(a+1) = %d\n", i, *(a+1));
21    s = malloc(sizeof(b));
22    memmove(s, b, sizeof(b));
23    for(sz=0, i=ALG; sz < 10; i++)
24      printf("f1(%d) = %d\n", i % 10, f1(&s, &sz, i % 10));
25    free(s);
26    free(a);
27    return 0;
28  }
```

a) [1] Considere para esta alínea que a instrução `printf` da linha 5 não está comentada, e admita que o código da função `main` foi executado até à linha 18, inclusive. Apresente, justificando, o que foi produzido em *standard output* e explique a funcionalidade da função **`f2`**.

b) [1] Admita que a execução do código da função **`main`** continuou até à linha 20, inclusive. Apresente, justificando, o que foi produzido em *standard output*. 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`**.

```
/*--------------------------------------------------------------------------------

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 dest, e fossem depois copiados para a memória apontada por dest.


Parâmetros:

void *dest: referência para a memória para onde é realizada a cópia.

const void *src: referência para a memória de onde é realizada a cópia.

size_t n: número de bytes a copiar.


Retorno:

void: não tem tipo de retorno.

--------------------------------------------------------------------------------*/
```

c) [1] Admita que a execução do código da função **`main`** continuou até à linha 25, exclusive. Apresente, justificando, o que foi produzido em *standard output* e represente, justificando com clareza, o conteúdo do *array* referenciado por **`s`** imediatamente antes da execução da linha 25.
Considerando que o compilador usa 4 bytes para alojar um `int`, estime quantos bytes de memória dinâmica serão libertados pela execução das linhas 25 e 26.

Nota: a ausência de justificação do conteúdo do *array* referenciado por **`s`** penalizará significativamente a avaliação desta alínea.

d) [1] Considere que se pretende agora realizar uma função **`f2a`** com a mesma funcionalidade da função **`f2`** mas sem recursividade. Sem utilizar qualquer função da biblioteca normalizada, escreva a definição da função **`f2a`**, adicionando ou removendo parâmetros se necessário. Escreva a definição da função com uma indentação correta, usando obrigatoriamente comentários.

2. [4 valores] Pretende-se fazer o registo de qualidade de produtos usando uma estrutura composta pelos seguintes campos:

```
typedef struct{
  char  *class;     // String com a classificação do produto (Ex: "A", "BX", "BAC")
  int    minValue;  // Menor valor admissível de acordo com o critério de qualidade
  int    maxValue;  // Maior valor admissível de acordo com o critério de qualidade
} NET_CLASS;
```

a) [1,5] Escreva a função

```
NET_CLASS *create_class(char *v_class, int v_min, int v_max);
```

que cria e preenche uma nova estrutura capaz de armazenar dados relativos a um registo de qualidade, tendo o cuidado de colocar em maiúsculas a nova *string* com a classificação a armazenar na estrutura, recorrendo obrigatoriamente a uma função da biblioteca normalizada, declarada em **`ctype.h`**.

b) [1] Escreva a função

```
int compare_class(NET_CLASS *v1, NET_CLASS *v2);
```

que compara a informação de qualidade de dois registos de dados retornando um valor negativo, zero ou um valor positivo, usando o seguinte critério pela ordem indicada:

- A ordem alfabética do campo **`class`** reflete a maior ou menor qualidade do produto (ex: "A" < "B");
- O desempate entre produtos de igual **`class`** é feito pelo valor da diferença entre **`maxValue`** e **`minValue`**.

Exemplos:

Classificação #1	Classificação #2	Resultado	Observações
"RX" | 1 | 3	"RX" | 5 | 7	0	"RX" é alfabeticamente igual a "RX" e (3-1) é igual a (7-5)
"ADRV" | 2 | 5	"AXC" | 1 | 4	<0	"ADRV" é alfabeticamente menor que "AXC"
"ABC" | 6 | 6	"ABC" | 8 | 9	<0	"ABC" é alfabeticamente igual a "ABC" e (6-6) é menor que (9-8)
"ABC" | 1 | 5	"ABC" | 6 | 9	>0	"ABC" é alfabeticamente igual a "ABC" e (5-1) é maior que (9-6)
"X" | 1 | 5	"ABC" | 6 | 9	>0	"X" é alfabeticamente maior que "ABC"

c) [1,5] Escreva a função

```
NET_CLASS *get_class_min(NET_CLASS v[], size_t n);
```

que devolve o endereço do elemento com menor classificação existente no *array* (não ordenado) de classificações **`v`** que contém **`n`** elementos, usando obrigatoriamente a função de comparação especificada na alínea anterior.

3. [5 valores] Pretende-se fazer o registo de produtos e respetiva qualidade usando uma estrutura composta pelos seguintes campos:

```
typedef struct product{
  char      *fullname;  // Nome completo do produto (brand + hífen (-) + model)
  NET_CLASS *eval;      // Avaliação do produto
  int       isNew;      // Valor lógico que indica se se trata de um novo produto
                        // 1=Novo Produto   0=Produto antigo
} PRODUCT;
```

a) [1,5] Escreva a função

```
PRODUCT *create_product(char *brand, char *model, char *class,
                        int par_min, int par_max);
```

que cria e preenche uma nova estrutura capaz de armazenar os dados relativos a um novo produto e respetivo registo de qualidade, recorrendo obrigatoriamente à função que cria a classificação do produto. De realçar que o campo **`fullname`** será criado dinamicamente com o conteúdo dos parâmetros **`brand`** e **`model`** separados por um hífen ('**`-`**').

b) [1,5] Escreva a função

```
int compare_product(PRODUCT *p1, PRODUCT *p2);
```

que compara dois registos de produtos retornando um valor negativo, zero ou um valor positivo, recorrendo obrigatoriamente à função que faz a comparação de classificações especificada no exercício anterior, usando o seguinte critério pela ordem a seguir indicada:

- Registos novos devem aparecer antes de registos mais antigos;
- Critério de qualidade;
- Ordem alfabética do campo **`fullname`**.

c) [2] Escreva a função

```
PRODUCT **getRefSortedArray(PRODUCT *arr, size_t count);
```

que aloja dinamicamente, preenche, ordena e retorna um *array* de ponteiros para os elementos do *array* indicado pelo parâmetro **`arr`**. O parâmetro **`count`** indica a quantidade de elementos do *array*. Para realizar a ordenação deve usar a função **`qsort`**, usando obrigatoriamente a função de comparação `compare_product`.

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

4. [4 valores] Pretende-se criar uma estrutura de dados para aceder aos elementos do tipo **`PRODUCT`**, definido no exercício 3, organizados em subconjuntos, os quais são agrupados por *brand*. Os produtos de cada *brand* são acedidos através de uma lista ligada que contém ponteiros para os elementos referenciados.

   Com vista a disponibilizar a pesquisa eficiente por *brand*, cria-se também uma árvore binária de pesquisa em que cada elemento contém o ponteiro para uma das listas ligadas que identificam os subconjuntos.

   Os descritores seguintes, **`LIST_EL`** e **`BST_EL`**, caraterizam os elementos das listas ligadas e da árvore binária.

```
typedef struct list_el{
    struct list_el *link;  // ponteiro de ligação na lista
    PRODUCT *prod;         // descritor de produto referenciado
} LIST_EL;

typedef struct bst_el{
    struct bst_el *lLink;  // ponteiro para a subárvore esquerda
    struct bst_el *rLink;  // ponteiro para a subárvore direita
    char *brand;           // string com a brand do subconjunto referenciado
    LIST_EL *list;         // ponteiro para a lista ligada com as referências do subconjunto
} BST_EL;
```

a) [2] Escreva a função

```
LIST_EL *listAddRef( LIST_EL *head, PRODUCT *prod );
```

que adiciona a uma lista ligada a referência para o descritor de um produto. Os parâmetros representam: **`head`**, o ponteiro cabeça da lista; **`prod`**, o descritor a referenciar. A lista é ordenada, utilizando obrigatoriamente a função **`compare_product`** especificada no exercício 3, alínea b). A função **`listAddRef`** retorna o endereço do primeiro elemento da lista.

b) [2] Escreva a função

```
BST_EL *bstAddRef( BST_EL *root, char *brand, PRODUCT *prod );
```

que adiciona à estrutura de dados a referência para o descritor de um produto. A árvore é ordenada alfabeticamente pelo campo **`brand`** dos seus elementos; a inserção de novos elementos é realizada nas folhas. Os parâmetros representam: **`root`**, o ponteiro raiz da árvore; **`brand`**, a identificação do subconjunto a selecionar (ou criar, se necessário); **`prod`**, o descritor a referenciar. A função **`bstAddRef`** retorna o endereço do elemento na raiz da árvore.

5. [3 valores] Considere um conjunto de módulos escritos em linguagem C, **`eval.c`**, **`shift.c`**, **`match.c`** e **`find.c`**, compilados individualmente com o comando "**`gcc -c *.c`**".

   Na caixa abaixo apresenta-se 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*.

```
eval.o:
0000000000000000 T eval

find.o:
0000000000000000 T find
                 U match
                 U shift

match.o:
0000000000000000 T match

shift.o:
0000000000000000 T shift
```

   Considere também que há um módulo de aplicação, **`prog.c`**, contendo a função **`main`** seguinte

```
int main(int argc, char **argv ){
    if( argc != 3 ){
        fprintf( stderr, "Usage: %s pattern size\n", argv[0] );
        return EXIT_FAILURE;
    }
    int p = strtol( argv[1], NULL, 10 ), s = strtol( argv[2], NULL, 10 );
    int x;
    while( scanf( "%d", &x ) == 1 ){
        int r = find( p, s, x );
        if( r != -1 ){
             printf( "Pattern found at position %d\n", r );
        }
    }
    return EXIT_SUCCESS;
}
```

   Admita que são criados, com vista à utilização no processo de compilação, os *header files* **`eval.h`**, **`shift.h`**, **`match.h`** e **`find.h`**, cada um contendo as assinaturas das funções públicas do módulo fonte (**`.c`**) com o respetivo prefixo.

   a) [0,5] Sabendo que o módulo **`prog.c`** é compilado com o comando "**`gcc -c prog.c`**", apresente a lista de símbolos produzida por "**`nm prog.o`**" e a respetiva classificação (**`t`**, **`T`** ou **`U`**).

   b) [0,5] Diga, justificando, se há funções definidas com o atributo **`static`** e, se houver, quais são.

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

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