Instituto Superior de Engenharia de Lisboa

Programação II

2022/23 – 2.º semestre letivo

Teste de época de recurso

2023.07.22

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

1. [5 valores]

Considere o troço de código constante da caixa seguinte. Na linha 1 daquele troço de código 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   int show(int *e, int * p){
6     printf("%d ", *e);
7     return 1;
8   }
9   int f2(const int * e) {
10    return (*e == ALG);
11  }
12  int chg(int * n, int * lim){
13    int r = (*lim < 5);
14    *n = r ? *n * *n : -*n;
15    return r;
16  }
17  int f1( int a[], size_t size, int (*cond)(const int * e),
18          int (*act)( int *data, void *context ), void * context ) {
19    size_t i, s=0;
20    for(i = 0; i < size; i++)
21      if(cond == NULL || cond(&a[i]))
22        s += act(&a[i], context);
23    return s;
24  }
```

a) [1,5] No troço de programa seguinte consta um exemplo de utilização da função `f1`.

```
int *a, option=ALG, r;
(…) // Preenchimento do array dinâmico a.
size_t size=20;
r = f1(a, size, NULL, show, NULL);
```

Admita que o conteúdo do *array* **`a`** antes da invocação da função `f1` é o seguinte:

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

Indique, justificando, qual é o *standard output* produzido pela execução deste troço de código e apresente, justificando, o conteúdo da variável **`r`**.

b) [1,5] No troço de programa seguinte consta outro exemplo de utilização da função f1.

```
int *a, option=ALG, r;
(…) // Preenchimento do array dinâmico a.
size_t size=20;
r = f1(a, size, f2, chg, &option);
```

Apresente, justificando de forma detalhada, o conteúdo da variável **r** e do *array* **a** após a execução do troço de código anterior, admitindo que o conteúdo daquele *array* antes da invocação da função `f1` é o indicado na alínea anterior.

Nota: a ausência de justificação do conteúdo do *array* **a** (após a execução do troço de código anterior) penalizará significativamente a avaliação desta alínea.

c) [2] No troço de programa seguinte consta ainda outro exemplo de utilização da função f1.

```
int *a, option=ALG, r;
(…) // Preenchimento do array dinâmico a.
size_t size=20;
r = f1(a, size, f3, f4, &option);
```

Pretende-se que a execução da função `f1` opere a seguinte transformação no *array* **a**: todos os elementos com números ímpares deverão passar a conter o valor `ALG`.

Apresente, justificando, o código das funções `f3` e `f4` que realizam aquela transformação, tendo em conta que após a execução de `f1` a variável **r** deverá ter o valor 10.

Nota: desaconselha-se veementemente a escrita da definição destas funções sem a respetiva justificação porque penalizará significativamente a avaliação desta alínea.

2. [6 valores]

Pretende-se implementar uma estrutura de dados para armazenar conjuntos de *strings* codificadas em ASCII básico.

Considere o armazenamento das palavras em listas ligadas formadas por elementos com o tipo `Elem`. Pretende-se favorecer a rapidez da inserção na lista. Não se pretende dispor de acesso ordenado às *strings*.

```
typedef struct elem {
    struct elem *next;   // ligação em lista
    char *str;           // string armazenada; alojar dinamicamente
} Elem;
```

a) [2] Escreva a função

```
Elem *store( Elem *h, char *s );
```

que adiciona a uma lista, cujo primeiro elemento é indicado por `h`, uma réplica alojada dinamicamente da *string* indicada por `s`. A função retorna o novo ponteiro para o início da lista. Assuma, por simplificação, que nunca ocorre falta de memória para alojamento dinâmico.

b) [2] Escreva a função

```
int find( Elem *h, char *s );
```

que procura na lista, cujo primeiro elemento é indicado por `h`, uma *string* idêntica à indicada por `s`. Retorna: 1, se existe; 0, se não existe.

c) [2] Escreva a função

```
void delete( Elem *h );
```

que elimina a lista ligada indicada por `h`, libertando toda a memória de alojamento dinâmico que ela usa.

3. [5 valores]

Aproveitando a utilização das funções de listas ligadas, especificadas no grupo de exercícios anterior, pretende-se implementar o armazenamento de *strings* organizado em subconjuntos. Para definir os subconjuntos, as *strings* são agrupadas pelo primeiro caráter de cada uma delas. Assim, cada subconjunto contém *strings* com o mesmo caráter no início.

Os subconjuntos são implementados em listas ligadas com a especificação anterior, acedidas através de uma árvore binária de pesquisa formada por nós com o tipo `StrSet`, ordenada pelo código ASCII do caráter que identifica cada subconjunto, com os menores ligados pelo campo `lower`.

```
typedef struct strSet {
    struct strSet *lower, *higher; // ligações na árvore
    Elem *list;                    // lista com o subconjunto
} StrSet;
```

Na implementação das funções pretendidas, pode usar outras das especificadas, bem como escrever funções auxiliares. Valoriza-se a eficiência.

a) [2] Escreva a função

```
StrSet *place( StrSet *r, char *s);
```

que se destina a colocar, na árvore identificada pela raiz `r`, uma réplica da *string* indicada por `s`. Esta deve ser inserida no subconjunto selecionado pelo seu primeiro caráter. Se o subconjunto não existir, deve ser criado. Deve verificar se a *string* já existe, de modo a evitar o armazenamento repetido. A função retorna a raiz da árvore, eventualmente modificada pela inserção. Assuma, por simplificação, que nunca ocorre falta de memória para alojamento dinâmico.

b) [1,5] Escreva a função

```
int search( StrSet *r, char *s);
```

que se destina a procurar, na árvore identificada pela raiz `r`, uma *string* idêntica à indicada por `s`. Retorna: 1, se existe; 0, se não existe.

c) [1,5] Escreva a função

```
void deleteAll( StrSet *r );
```

que elimina a árvore binária identificada pela raiz `r`, libertando toda a memória de alojamento dinâmico que ela usa.

4. [4 valores]

Considere um conjunto de módulos escritos em linguagem C, compilados individualmente com o comando "`gcc -c m_*.c`".

Na caixa ao lado apresenta-se, relativamente aos de módulos compilados, a lista de símbolos, obtida com o comando "`nm m_*.o`", cuja classificação é: `T`, *public text*; `t`, *private text*; `U`, *undefined*.

```
m_a.o:
00000000000002e T fa
                U fb
0000000000000000 t fx

m_b.o:
0000000000000013 T fb
                U fc
0000000000000000 t fy

m_c.o:
0000000000000000 T fc

m_d.o:
                U fb
0000000000000000 T fd
```

a) [1] Indique, justificando, quais dos módulos fonte (.c) contêm funções definidas com o atributo `static`.

b) [1,5] Pretende-se dispor de vários *header files*, um por cada módulo dos referidos. Indique, para cada um dos *header files*: quais as funções cujas assinaturas deve conter; quais os módulos fonte em que deve ser incluído. Supondo que todas as funções têm dois parâmetros com tipo `int` e retorno com tipo `int`, escreva completamente o *header file* `m_b.h`, com as declarações necessárias e o controlo da inclusão múltipla.

c) [1,5] Admitindo que existe um módulo de aplicação, `prog.c`, que usa a função `fd`, escreva um *makefile* para gerar, de forma eficiente, o respetivo executável com o nome `prog`. Identifique, dos símbolos listados na caixa, quais existem no executável `prog` e a respetiva classificação, com a simbologia usada pelo comando `nm`.