Instituto Superior de Engenharia de Lisboa

Programação II

2021/22 – 1.º semestre letivo

Teste de época normal

2022.02.07

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 seguintes funções:

```
typedef struct{
  char *word;
  int freq;
} WordFreq;

int f2(WordFreq * wf1, WordFreq * wf2) {
  return !strcmp(wf1->word, wf2->word);
}

WordFreq ** f1( WordFreq * a[], size_t *size,
                int (*cond)(WordFreq *, WordFreq *), WordFreq * key,
                void (*act)(WordFreq *)) {
  size_t i = *size;
  while(i--)
    if(cond(a[i], key)) {
      if(act) act(a[i]);
      memmove(&a[i], &a[i+1], (--*size - i) * sizeof(a[0]));
      printf("%zu %zu\n", i, *size);
    }
  return realloc(a, *size * sizeof (a[0]));
}
```

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

```
int main() {
  WordFreq ** a, ** b;
  char * aa[] = {"sara", "ana", "nuno", "sara", "ana", "ana"};
  size_t i, size = sizeof(aa)/sizeof(aa[0]);
  a = malloc(size * sizeof(*a));
  for(i=0; i<size; i++){
    a[i] = malloc(sizeof(**a));
    a[i]->word = strdup(aa[i]);
  }

  WordFreq tmp = {"ana", -1};
  b = f1(a, &size, f2, &tmp, NULL);
  for(i=0; i<size; i++){
    printf("%s\n", b[i]->word);
    free(b[i]->word);
    free(b[i]);
  }
  free(b);
  return 0;
}
```

a) [1] Apresente os valores produzidos em *standard output* resultantes da execução do troço de código anterior, justificando. 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.
----------------------------------------------------------------------------*/
```

b) [1,5] Para o caso exemplificado no troço de código anterior, indique:

1) Quantos blocos de memória dinâmica (blocos no `heap`) são requisitados/alocados e qual o tamanho esperado, em `bytes`, de cada bloco?
2) Quantas vezes é chamada a função `free` e explique que bloco de memória é libertado em cada chamada.
3) Se toda a memória dinâmica requisitada foi libertada. Se foi, refira se podia ter sido libertada com menos chamadas à função `free`. Se não foi, refira, sem adicionar mais instruções à função `f1`, como podia ter sido libertada.

c) [1,5] 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 ponteiros para outros tipos e não apenas *arrays* de ponteiros para estruturas `WordFreq`. Escreva a definição da função `f1a` e, se necessário, adicione ou remova parâmetros. Escreva a função com uma indentação correta, usando <u>obrigatoriamente</u> comentários para que fique clara a sua implementação/funcionalidade.

2. [3,5 valores] Pretende-se fazer o registo de viaturas presentes no interior de uma área restrita, usando um *array* de elementos do tipo `Car`, alojado dinamicamente. O *array* é gerido através de um descritor com o tipo `CarSet`.

```
#define CAR_PLATE_SIZE 10

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 alojados e preenchidos no array cars
    Car *cars; // array alojado dinamicamente
} CarSet;
```

Pretende-se dispor de funções para: adicionar uma matrícula ao conjunto; marcá-la como presente na área; marcá-la como ausente; verificar se está presente.

a) [1] Escreva a função

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

que regista a entrada na área associada ao conjunto `set`, ativando o estado presente para a matrícula `plate`. Se a matrícula ainda não existir é inserida, usando a função `realloc` de biblioteca para adicionar um elemento ao *array*.

b) [1] Escreva a função

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

que regista a saída da área associada ao conjunto `set`, desativando o estado presente para a matrícula `plate`. A função normalmente retorna 1; se matrícula não existir, retorna 0.

c) [1] Escreva a função

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

que verifica se a matrícula `plate` está registada como presente na área associada ao conjunto `set`. Retorna 1, se está presente, ou 0, no caso de não existir ou estar marcada como ausente.

d) [0,5] Diga se considera possível melhorar a eficiência da função `carVerify` que apresentou. Se considerar que não, justifique; se considerar que sim, indique que partes do programa modificaria e descreva resumidamente essas modificações.

3. [5 valores] Pretende-se construir um modulo genérico para uma estrutura de dados do tipo *Stack* implementada à custa de uma lista ligada. Um *Stack* é uma estrutura de dados do tipo *LIFO* (*Last In First Out*), com duas operações típicas:

- **Push** – adiciona um elemento à estrutura
- **Pop** – remove o último elemento inserido na estrutura

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

a) [1] Escreva a função

```
void stackPush(StackNode **stack, void *data);
```

que coloca, no *Stack*, o elemento `data`, previamente alojado e preenchido.

b) [1] Escreva a função

```
void *stackPop(StackNode **stack);
```

que retorna o elemento mais recentemente colocado no *Stack* e elimina-o do mesmo. A função retorna `NULL` se o *Stack* estiver vazio.

c) [2] Escreva a função

```
StackNode* stackFind(StackNode* stack, void *data, int(*cmp)(void*,void*));
```

que retorna o nó cujo conteúdo seja igual ao de `data`, ou `NULL` caso contrário. A função `cmp` retorna -1, 0 ou +1 se o valor do primeiro parâmetro é, respetivamente, inferior, igual ou superior ao do segundo.

d) [1] Escreva a função

```
void stackDestroy(StackNode* stack);
```

que liberta a memória utilizada nos nós para construir a lista ligada que implementa o *Stack*.

4. [5 valores] Pretende-se implementar uma estrutura de dados, baseada numa árvore binária de pesquisa, onde seja possível guardar a informação de palavras incluídas numa dada linha de um ficheiro. Para tal, considere a seguinte declaração:

```
struct m_tnode{
   int line;                  //linha do ficheiro
   StackNode* words;          //stack com as palavras
   struct m_tnode *left, *right;  //ponteiros de ligação na árvore
}TNode;
```

A árvore construída para esta estrutura de dados deve estar ordenada pelo número de linha.

a) [1] Escreva a função

```
void lineAdd(TNode **root, int line);
```

que adiciona uma nova linha à estrutura de dados, se ainda não existir o número `line`.

b) [2] Escreva a função

```
void lineAddWord(TNode **root, int line, char *word);
```

que adiciona uma nova palavra ao *Stack* de uma dada linha. Se a linha ainda não existir, deve adicioná-la. A palavra `word` é previamente criada, em alojamento dinâmico controlado pelo programa de aplicação.

c) [1] Escreva a função

```
int lineCount(Tnode* c);
```

que retorna quantas linhas estão representadas na estrutura de dados.

d) [1] Escreva a função

```
void lineDestroy(TNode* root);
```

que liberta toda a memória usada para construir a estrutura de dados.

5. [2,5 valores] Considere o programa de aplicação `aplic.c` e as listas de símbolos obtidas, com a ferramenta a `nm`, a partir dos módulos compilados referidos no quadro seguinte.

`aplic.c`

```
#include <stdio.h>
#include <stdlib.h>

#include "datastore.h"

int main( int argc, char *argv[] ){
    Store_t *s =
      data_read_and_store( argc, argv );
    if( s == NULL ){
        fprintf( stderr,
          "Fail reading %s\n", argv[1] );
        exit( -1 );
    }
    data_list_all( s );
    data_delete( s );
    return 0;
}
```

`nm dataelement.o datastore.o`

```
dataelement.o:
0000000000000000 t elementAlloc
0000000000000014 T elementCreate
0000000000000046 T elementDelete
                 U free
                 U malloc
                 U strcpy

datastore.o:
                 U calloc
000000000000019e T data_delete
000000000000013a T data_list_all
000000000000006d T data_read_and_store
0000000000000000 T data_store
                 U element_create
                 U element_delete
                 U fgets
                 U fopen
                 U free
                 U printf
                 U puts
                 U realloc
```

Admita que não se dispõe do código fonte dos módulos `dataelement.o` e `datastore.o` existindo apenas o código compilado e os *header files* seguintes, com os conteúdos indicados:

`dataelement.h`, contém a definição do tipo `Elem_t` e as assinaturas de funções de `dataelement.o`;

`datastore.h`, contém a definição do tipo `Store_t` e as assinaturas de funções de `datastore.o`.

a) [0,5] Indique, justificando, se há, nos módulos `dataelement.o` e `datastore.o`, funções em cuja definição tenha sido usado o atributo `static`. Se houver, identifique-as.

b) [0,5] Admitindo o uso do comando "`gcc -c aplic.c`" para gerar o módulo compilado `aplic.o`, apresente a respetiva lista de símbolos, produzida pela ferramenta `nm`, indicando os nomes dos símbolos e a correspondente classificação, `t` (*private code*), `T` (*public code*) ou `U` (*undefined*). Indique, por cada símbolo indefinido, qual é o módulo que o resolve ou, em alternativa, se é resolvido pela biblioteca normalizada.

c) [0,5] Note que, nos módulos compilados fornecidos, há nomes escritos com dois estilos diferentes: palavras separadas por *underline* ou palavras destacadas com maiúscula. Sabendo que a tentativa de gerar o executável com o comando "`gcc aplic.c dataelement.o datastore.o`" não tem sucesso, descreva o motivo e indique se ele ocorre na fase de compilação ou na fase de ligação.

d) [1] Pondere a possibilidade de gerar o executável com o auxílio de um módulo adicional. Se considerar impossível, apresente o motivo; se considerar possível, escreva o novo módulo e um exemplo de comando(s) adequado(s) para produzir o executável. Considere as assinaturas seguintes, pertencentes ao *header file* `dataelement.h`:

```
Elem_t *elementCreate( char *line );

void elementDelete( Elem_t *e );
```