Instituto Superior de Engenharia de Lisboa

Programação II

2022/23 – 1.º semestre letivo

Teste de época normal

2023.01.12

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

1. [6 valores]

Para a realização da Série de Exercícios 3 foi utilizado um módulo para armazenamento da localização de linhas no exercício 2. De acordo com o enunciado, a descrição desse módulo era a seguinte:

> Pretende-se armazenar a localização de linhas de texto arbitrárias, pertencentes a vários ficheiros.
>
> Assume-se que os nomes dos ficheiros estão armazenados em *strings* acessíveis, permanentemente, através de um *array* de ponteiros. O tipo `Location` seguinte destina-se a registar a localização de uma linha; o significado dos campos é descrito nos comentários. Os elementos deste tipo são agrupados em *arrays* dinâmicos (vetores, com o tipo `VecLoc`) que representam as localizações de conjuntos de linhas.

```
typedef struct{     // Descritor de uma localização de linha
  int fileIdx;                  // índice no array de ponteiros para os nomes dos ficheiros
  int line;             // número da linha no texto
  long offset;          // posição do início da linha no ficheiro
} Location;

typedef struct{     // Descritor de um vetor de localizações de linhas
  int space;            // quantidade de elementos alojados no array dinâmico
  int count;            // quantidade de elementos ocupados no array dinâmico
  Location *data;       // aponta array alojado dinamicamente
} VecLoc;
```

Antes de iniciar a realização deste grupo, leia o enunciado de todas as alíneas que o compõem.

No desenvolvimento das funções seguintes deve ter em conta que o vetor de localizações, passado no parâmetro `vec`, está sempre ordenado e deve manter-se ordenado. A ordem é crescente por `fileIdx`; para o mesmo `fileIdx`, é crescente por `line`. O vetor não contém repetições, isto é, nas localizações relativas a um dado ficheiro, não existe mais do que uma de cada linha.

a) [1] Escreva a função

```
int count_line_in_files(VecLoc *vec, int line);
```

que retorna a contagem de ficheiros que contêm alguma localização com o número de linha passado em `line`.

b) [2] Escreva a função

```
void delete_location(VecLoc *vec, int loc_idx);
```

que remove do vetor de localizações, passado em `vec`, a localização com índice `loc_idx`, mantendo o vetor ordenado.

c) [3] Escreva a função

```
int insert_location(VecLoc *vec, Location *loc);
```

que se destina a inserir no vetor, indicado por `vec`, um novo elemento com os dados passados em `loc`, mantendo o vetor ordenado e sem repetições. Se já existir um elemento igual, não se realiza qualquer modificação. A função deve assegurar espaço no vetor, alojado dinamicamente, para o novo elemento; se necessário, deve redimensionar o espaço, fazendo-o em blocos de 10 elementos, com recurso à função `realloc` de biblioteca.

A função `insert_location` retorna: `0`, caso a localização indicada por `loc` já exista no vetor; `1`, em caso de sucesso na inserção; `-1`, em caso de insucesso por falta de memória de alojamento dinâmico. Note que, caso não haja memória dinâmica disponível, o vetor deve ser mantido inalterado.

Sugere-se que percorra o vetor sequencialmente para identificar o ponto de inserção do novo elemento. Neste exercício não deve utilizar a função `qsort` nem a função `bsearch` da biblioteca normalizada.

2. [6 valores]

Considere a seguinte definição de um nó de uma lista ligada, que representa as ocorrências de uma palavra num ficheiro:

```
typedef struct lNode{
   struct lNode *next; // ponteiro de ligação da lista
   char *word;         // string, alojada dinamicamente, representando uma palavra
   int freq;           // a frequência da palavra num ficheiro
} LNode;
```

Admita que a lista não está sujeita a nenhum critério de ordenação.

a) [1] Escreva a função

```
LNode* findWord( LNode* head, char* word );
```

que retorna o nó da lista que contém a palavra `word`, ou `NULL` caso não exista.

b) [2] Escreva a função

```
LNode* addWord( LNode* head, char* word );
```

que adiciona uma nova palavra à lista, caso esta ainda não esteja presente. Caso já exista, incrementa o número de ocorrências da mesma. A função retorna a cabeça da lista.

c) [3] Escreva a função

```
LNode* mergeLists( LNode* list1, LNode* list2 );
```

que retorna uma **nova** lista com as mesmas palavras contidas em `list1` e `list2`. A nova lista deve ter as palavras provenientes das duas listas originais, sem repetição; cada palavra deve ter a soma das respetivas ocorrências em ambas as listas. Serão valorizadas as soluções que tenham em conta o desempenho da execução.

3. [4 valores]

Pretende-se desenvolver um programa para catalogar ficheiros de áudio com obras musicais, onde se regista, por cada obra, o artista principal, o título, a duração em minutos e o nome do ficheiro. A estrutura de dados é uma árvore binária de pesquisa, com nós do tipo `TNode`, identificando uma obra em cada nó e tendo ordenação alfabeticamente crescente, da esquerda para a direita, pelo nome do artista; se houver várias obras do mesmo artista, a posição relativa entre elas é ordenada pelo título.

```
typedef struct tNode{
   struct tNode *left, *right; // ponteiros de ligação na árvore
   char artist[MAX_ARTIST+1];  // string, nome do artista
   char title[MAX_TITLE+1];    // string, título da obra
   char *filename;             // string, alojamento dinâmico, nome do ficheiro
   double duration;            // duração da obra em minutos
} TNode;
```

a) [1] Escreva a função

```
int tPrintByDuration( TNode *r, double dur );
```

que apresenta, em *standard output*, o artista, título, duração e ficheiro das obras com duração superior a `dur`. Os dados devem ser apresentados pela ordem crescente da árvore. Retorna a quantidade de obras apresentadas.

b) [2] Escreva a função

```
int tPrintByArtist( TNode *r, char *art );
```

que apresenta, em *standard output*, o título, duração e ficheiro das obras do artista indicado por `art`. Os dados devem ser apresentados pela ordem crescente da árvore. Retorna a quantidade de obras apresentadas. Valoriza-se a eficiência, devendo ser evitados percursos em partes desnecessárias da árvore.

c) [1] Escreva a função

```
void tDelete( TNode *r );
```

que elimina a árvore com raiz indiada por `r`, libertando a memória de alojamento dinâmico usada nos seus nós e nas *strings* dependentes deles.

4. [4 valores]

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

```
comp1.o:
0000000000000000 T comp1

comp2.o:
                 U comp1
0000000000000000 T comp2

sortint.o:
0000000000000000 t select
0000000000000057 T sortInt
                 U swap

swap.o:
                 U memcpy
0000000000000000 T swap
```

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

```
int main(){
     int a[ARR_SIZE];
     int c = 0;
     int (*comp_func)(int *, int*);
     comp_func = comp1;
     while( c < ARR_SIZE && scanf( "%d", a + c ) == 1 )
       ++c;
     sortInt( a, c, comp_func );
     for( int i = 0; i < c; ++i )
       printf( "%d\t", a[i] );
     putchar( '\n' );
     return 0;
}
```

Admita que são criados e usados no processo de compilação os *header files* `comp1.h`, `comp2.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 test.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, justificando, se nos módulos referidos: (1) Há funções com o atributo `static`? (2) Há funções a que se possa adicionar o atributo `static`?

c) [1] Escreva o *header file* `sortint.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* `sortint.h` deve ser incluído.

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