Instituto Superior de Engenharia de Lisboa

LEIRT – LEETC

Programação II

2021/22 – 2.º semestre letivo

Terceira Série de Exercícios

Pretende-se uma nova implementação do programa especificado na segunda série de exercícios, com as modificações seguintes:

- Exploração de alojamento dinâmico de memória;
- Organização do código em módulos separados.

O alojamento dinâmico é utilizado para conter os *arrays* utilizados em diversas situações, em vez da sua declaração direta como variáveis de dimensão fixa.

A organização em módulos visa estruturação independente de funcionalidades específicas como, por exemplo, a leitura do ficheiro, a separação das palavras ou a gestão dos dados armazenados. As funções anteriormente desenvolvidas, e que continuem a ser úteis, devem ser reutilizadas e integradas na organização em módulos referida.

Tal como especificado na série de exercícios anterior, o programa recebe, como argumento de linha de comando, o nome de um ficheiro de texto contendo a tabela de dados. Na sua atividade há duas fases distintas – preparação de dados e execução de comandos. Na fase de preparação de dados, lê a tabela e preenche a estrutura de dados. Na fase de execução de comandos, apresenta, ciclicamente, um *prompt* e espera, de *standard input*, um dos comandos seguintes.

- `a` – (*artists*) apresenta a lista de faixas áudio ordenada por *Artist*. Se houver várias faixas do mesmo intérprete, aplica, sucessivamente, a ordem por *Album* e *Title*;
- `t` – (*titles*) apresenta a lista de faixas áudio ordenada por *Title*. Se houver várias faixas com título idêntico, aplica, sucessivamente, a ordem por *Artist* e *Album*;
- `s` *`title`* – (*search*) apresenta a faixa áudio com o título indicado pelo parâmetro. Se o título não existir, apresenta uma mensagem de erro;
- `q` – (*quit*) termina.

As listas produzidas pelos comandos "`a`" e "`t`", em *standard output*, são por ordem alfabética crescente. Cada linha de texto em *standard output*, bem como a linha de resposta ao comando "`s`", exibe a informação de uma faixa áudio, contendo os campos da *tag* separados por ponto-e-vírgula.

1. Aspetos essenciais das estruturas de dados dinâmicas

Propõe-se que se mantenha a utilização do tipo `MP3Tag_t` definido na série de exercícios anterior. No entanto, em vez de criar um *array* de elementos deste tipo, pretende-se que as estruturas `MP3Tag_t` sejam criadas individualmente, em alojamento dinâmico, e referenciadas através de *arrays* de ponteiros, também com alojamento dinâmico.

Esta forma tem o propósito de evitar a necessidade de criar um bloco de contíguo de dimensão muito elevada, em alojamento dinâmico, para o *array* de *tags*, e o consequente custo de realojamento, bem como uniformizar o acesso aos mesmos dados com diferentes critérios de ordenação.

Assim, não são usados os tipos `TagArr_t` e `TagRef_t` anteriores. Em substituição destes, utiliza-se agora um novo tipo, `DinRef_t`, que suporta a criação e crescimento dinâmico de *arrays* de ponteiros para os elementos `MP3Tag_t` criados individualmente.

```
typedef struct{
        int space; // Quantidade de elementos alojados no campo refs.
        int count; // Quantidade de elementos preenchidos no campo refs.
        MP3Tag_t **refs; // Ponteiro para array de ponteiros para tag; alojamento dinâmico
} DinRef_t;
```

No novo tipo `DinRef_t` o campo `refs` é um ponteiro, destinado a apontar para um *array* de ponteiros, de modo a permitir o seu alojamento e realojamento dinâmico, para se adaptar automaticamente à quantidade de *tags* existente. O campo `space` serve para controlar a quantidade de elementos alojados, de modo a permitir o realojamento por blocos com o objetivo de evitar o custo de um realojamento por cada elemento adicionado.

Tendo em conta a necessidade acesso às *tags* com dois critérios de ordenação diferentes, um para o comando "`a`"; outro para os comandos "`t`" e "`s`", propõe-se a utilização de dois *arrays* dinâmicos do tipo `DinRef_t`. Quando se cria cada *tag*, adiciona-se a sua referência aos dois *arrays*; no final da preparação ordena-se cada um dos *arrays* de referências com um dos critérios.

Propõe-se ainda o tipo `Manage_t`, que reúne o acesso aos dois *arrays* de referências, para organizar o código de gestão dos dados.

```
typedef struct{
        DinRef_t *refA;   // Referências para o comando "a".
        DinRef_t *refT;   // Referências para os comandos "t" e "s".
} Manage_t;
```

2. Módulos propostos

Organize o programa em módulos, caraterizando a funcionalidade e definindo as assinaturas das funções de interface de cada módulo. Propõe-se que considere os módulos indicados abaixo; no caso de os alunos terem implementado a série anterior em módulos, podem manter a respetiva organização e adicionar ou substituir o que for necessário, desde que a organização resultante seja adequada para implementar a aplicação pretendida e não implique a integração de código desnecessário no executável.

2.1. Módulo de *arrays* dinâmicos de referências; deve conter, pelo menos, as funções:

```
DinRef_t *dinRefCreate( int initSpace );
```

Esta função cria, em alojamento dinâmico, e retorna, um descritor de um *array* dinâmico de ponteiros para *tags*, alojando a quantidade de elementos indicada por `initSpace`. O descritor é iniciado no estado vazio.

```
void dinRefDelete( DinRef_t *ref );
```

Esta função elimina o *array* dinâmico de referências, indicado por `ref`. Deve libertar a memória dinamicamente alojada para o descritor e para o *array*, mas não a das *tags* referenciadas. Estas podem estar associadas a vários *arrays* de referências, pelo que não pertencem a nenhum deles, devendo ser eliminadas explicitamente. Para isso, pode ser usada a função `dinRefScan` especificada abaixo.

```
void dinRefAdd( DinRef_t *ref, MP3Tag_t *tag );
```

Esta função adiciona ao descritor, indicado por `ref`, o ponteiro para a estrutura indicada por `tag`, previamente alojada e preenchida. Quando necessário, redimensiona o *array*, utilizado a função `realloc` de biblioteca. O redimensionamento deve ser realizado em blocos de vários elementos.

```
void dinRefSort( dinRef_t *ref,
                 int (*compar)( const void *, const void* ) );
```

Esta função ordena o *array* de referências, pertencente ao descritor indicado por `ref`, com o critério implementado pela função de comparação passada em `compar`, compatível com a especificação da função `qsort`, de biblioteca, que deve ser usada para ordenar o *array* de referências.

```
MP3Tag_t *dinRefSearch( dinRef_t *ref, void *key,
                        int (*compar)( const void *, const void* ) );
```

Esta função realiza pesquisa binária no *array* de referências, pertencente ao descritor indicado por `ref`, aplicando a chave de pesquisa `key`, com o critério implementado pela função de comparação passada em `compar`, compatível com a especificação da função `bsearch`, de biblioteca, que deve ser usada para a pesquisa. A função retorna o endereço da *tag* encontrada ou NULL no caso de insucesso. Assume-se que o *array* foi previamente ordenado.

```
void dinRefScan( DinRef_t *ref, void (*action)( MP3Tag_t * ) );
```

Esta função percorre o *array* de referências, pertencente ao descritor indicado por `ref`, do índice 0 para o mais alto, aplicando a função passada em `action` a cada uma das *tags* referenciadas. Esta função é útil, por exemplo, para apresentar os dados das *tags* em *standard output* ou para libertar a memória ocupada pelas *tags*.

2.2. Módulo de gestão dos dados; deve conter, pelo menos, as funções:

```
Manage_t *manCreate( void );
```

Esta função aloja dinamicamente, e retorna, o descritor para gestão dos dados. Deve utilizar o módulo de *arrays* de referências para criar os respetivos descritores.

```
void manDelete( Manage_t *man );
```

Esta função elimina o descritor de gestão de dados, indicado por `man`, libertando a toda a memória de alojamento dinâmico que dele depende. Deve utilizar o módulo de *arrays* de referências para eliminar os respetivos descritores.

```
void manAddTag( Manage_t *man, MP3Tag_t *tag );
```

Esta função adiciona à gestão de dados, através do descritor indicado de `man`, a referência para a estrutura indicada por `tag`, previamente alojada e preenchida.

```
void manSort( Manage_t *man );
```

Esta função marca como concluída a fase de preparação dos dados, tendo sido criadas e referenciadas todas as *tags*. Tem como objetivo preparar os dois *arrays* de referências para dar as respostas pretendidas na fase de comandos. Para isso, aplica os respetivos critérios de ordenação, usando o módulo de *arrays* de referências.

```
void manCommand( Manage_t *man, char *cmdLine );
```

Esta função apresenta, em *standard output*, os resultados do comando inserido pelo utilizador. A linha de comando é passada através do parâmetro `cmdLine`. Deve utilizar as funções do módulo de *arrays* de referências para realizar as listagens e as pesquisas. O comando "`q`" é processado diretamente pela aplicação, não sendo chamada a função `manCommand`.

2.3. Módulo de leitura da tabela de dados; disponibiliza, pelo menos, a função:

```
int tableReadStore( char *tableName, Manage_t *man );
```

Esta função destina-se preencher a estrutura de dados *array* para acesso às *tags*. Deve percorrer o ficheiro de texto com o nome indicado por `tableName`, ler os dados de cada *tag* e adicionar a sua referência à gestão de dados, através do descritor indicado por `man`, usando a função `manAddTag`.

Tal como especificado na série anterior, no caso de haver carateres de espaço nas extremidades dos campos, estes espaços devem ser eliminados. Se houver *tags* em que um dos campos *Title* e *Artist* seja *string* vazia, a respetiva linha da tabela deve ser ignorada, não ficando registada para qualquer processamento.

Esta função retorna 0, em caso de sucesso, ou -1, em caso de falha na leitura da tabela.

2.4. Módulo de aplicação, responsável por gerir a obtenção e armazenamento dos dados, bem como a interação com o utilizador; invoca as funções dos módulos anteriores para:

- Iniciar o funcionamento da gestão dos dados;
- Percorrer a tabela recebida por parâmetro de linha de comando, obter os dados de cada *tag*, criar a sua representação e referenciá-la na gestão de dados;
- Acionar a passagem da fase de preparação à fase de comandos, organizando os acessos ordenados;
- Aceitar os comandos do utilizador e promover as respostas correspondentes.
- Finalizar a atividade de forma ordenada, promovendo nomeadamente a libertação da memória alojada dinamicamente, antes de terminar a execução.

3. Desenvolvimento

3.1. Escreva os módulos utilitários por fases, realizando sucessivos testes parciais para verificar o funcionamento de cada módulo na respetiva fase de desenvolvimento.

Tendo em conta a dependência de funcionalidades, o desenvolvimento deve ter a ordem seguinte:

1. Módulo de *arrays* de referências;
2. Módulo de gestão de dados;
3. Módulo de leitura da tabela de dados;

Em conjunto com cada módulo utilitário deve escrever o respetivo *header file*, contendo as declarações necessárias para a utilização do módulo, constando nomeadamente as assinaturas das funções de interface; no caso de definir funções auxiliares, estas devem ter *scope* privado.

Os testes parciais devem ser realizados com aplicações de teste específicas, produzidas por *makefiles* em versões adequadas a cada fase de desenvolvimento.

3.2. Desenvolva o módulo de aplicação final

Rescreva o código da aplicação final, adaptando a implementação anterior de modo a usar as funções de interface dos outros módulos.

Escreva o *makefile* para controlar, de forma eficiente, a produção do executável.

Teste o programa completo, reutilizando o ficheiro com a tabela de dados usado para ensaio da série anterior; pode criar outros ficheiros se considerar conveniente.

ISEL, 2022.05.20

(rev. 2022.05.24)