Instituto Superior de Engenharia de Lisboa

LEETC – LEIRT

Programação II

2019/20 – 2.º semestre letivo

Segunda Série de Exercícios

Esta série de exercícios consiste na implementação de um programa para catalogar um conjunto de faixas de áudio, armazenadas em ficheiros com formato MP3, com base na respetiva *tag* (formato *MPEG Audio Tag* ID3v1 ou ID3v1.1), permitindo listagem ordenada, com diferentes critérios aplicados aos seus campos de informação: *Title*; *Artist*; *Album*; *Year*; *Comment*; *Track* (só em ID3v1.1); *Genre*.

A *tag*, se existir, é armazenada num bloco de 128 bytes, colocado no final do ficheiro, após as *frames* de áudio e iniciada pela marca "TAG" em ASCII. As faixas que não disponham de *tag* são excluídas do processamento.

Além do estudo das normas (http://mpgedit.org/mpgedit/mpeg_format/mpeghdr.htm), propõe-se que complemente com a observação do conteúdo dos ficheiros MP3, por exemplo através do utilitário `hd`.

O programa recebe, como argumento de linha de comando, o nome de um ficheiro de texto contendo uma lista com os nomes dos ficheiros MP3 a catalogar. Após a leitura da lista e dos ficheiros que ela indica, ciclicamente, apresenta 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*;
- `q` – (*quit*) terminar.

As listas produzidas, em *standard output*, pela execução dos comandos são por ordem alfabética crescente. Cada linha de texto em *standard output* exibe a informação de uma faixa áudio, contendo os campos da *tag* e o nome do ficheiro MP3 correspondente.

1. Admita os tipos de estrutura `MP3Tag_t`, que representa a informação de uma *tag*, e `TagSet_t`, que representa o descritor de um conjunto de *tags*.

   Considere a organização do programa em módulos, existindo pelo menos os seguintes.

   1.1. Módulo `tag.c` – leitura dos ficheiros MP3; disponibiliza, pelo menos, a função

   ```
   MP3Tag_t *tagRead ( char *fileName, int *resPtr );
   ```

   Esta função lê a *tag* existente no ficheiro com o nome indicado por `fileName`, e armazena-a em memória alojada dinamicamente e retorna o seu endereço. O parâmetro `resPtr`, se for NULL deve ser ignorado; se não, indica uma variável que deve ser afetada com um código de resultado. Em caso de sucesso, a função retorna o endereço da *tag* preenchida e o código de resultado é 0; caso contrário, retorna NULL, não devendo alojar memória dinamicamente, e o código de resultado é -1, se falhar o acesso ao ficheiro, ou -2, se a faixa de áudio não tiver informação de *tag*.

   1.2. Módulo `setdata.c` – gestão dos dados, disponibilizando, pelo menos, as funções

   ```
   TagSet_t *setCreate( void );
   ```

   Esta função cria o descritor de conjunto de *tags* no estado inicial, pronto para adicionar a informação das tags. Retorna o endereço do descritor alojado dinamicamente.

```
void setAdd( TagSet_t *set, MP3Tag_t *tag );
```

Esta função adiciona ao descritor, indicado por `set`, uma *tag*, indicada por `tag`. Tendo em conta que o parâmetro `tag` indica um espaço alojado dinamicamente, a adição ao conjunto consiste em referenciá-lo num *array* de ponteiros para `MP3Tag_t`. Este *array* também é alojado dinamicamente e deve ser redimensionado quando necessário.

```
void setReady( TagSet_t *set );
```

Esta função marca como concluída a fase de inserção das *tags* e prepara o descritor do conjunto para dar as respostas pretendidas, na fase de comandos.

```
void setCommand( TagSet_t *set, char *cmdLine );
```

Esta função apresenta, em *standard output*, os resultados do comando inserido pelo utilizador. A linha de comando é passada à função através do parâmetro `cmdLine`.

```
void setDelete( TagSet_t *void );
```

Esta função, usada em consequência do comando "`q`", elimina o descritor de conjunto de *tags* e as estruturas de dados dependentes dele, libertando a respetiva memória de alojamento dinâmico.

1.3. Módulo `listfiles.c` – processamento da lista de ficheiros MP3, disponibilizando, pelo menos, a função

```
int listScan( char *listName, TagSet_t *data );
```

Esta função percorre o ficheiro de texto com nome indicado por `listName`, obtendo os nomes dos ficheiros MP3, armazenados um em cada linha de texto. Por cada ficheiro MP3, obtém a informação de *tag*, usando a função `tagRead`, e adiciona-a ao conjunto de *tags*, usando a função `setAdd`. Retorna 0, em caso de sucesso, ou -1, em caso de falha na leitura da lista.

Se ocorrer falha na obtenção da *tag* de alguma faixa, apresenta a ocorrência através de *standard error*, indicando o tipo de erro, e prossegue para a faixa seguinte; Não é armazenada nenhuma informação relativa às faixas de áudio em que falha a leitura da *tag*.

1.4. Módulo `app.c` – aplicação, responsável por gerir a obtenção e armazenamento dos dados, bem como a interação com o utilizador, invocando as funções dos módulos anteriores para:

- Iniciar o funcionamento da gestão dos dados;
- Percorrer a lista de ficheiros MP3, indicados na lista recebida por parâmetro de linha de comando, obter os dados da *tag* de cada um deles e transmiti-los ao módulo de gestão dos dados;
- Acionar a passagem da fase de inserção à fase de comandos;
- Aceitar os comandos do utilizador e promover as listagens correspondentes;
- Finalizar a atividade de forma ordenada, promovendo nomeadamente a libertação da memória alojada dinamicamente, antes de terminar a execução.

2. Desenvolva o código dos diversos módulos fonte. As funções internas dos módulos devem ter o atributo `static`, ficando como públicas apenas as funções de interface.

Escreva também os *header files*. Estes devem conter as assinaturas das funções públicas dos módulos respetivos módulos fonte, bem como definições de tipo quando aplicável; devem dispor de controlo da inclusão múltipla.

2.1. Tendo em conta a necessidade de manipular informação de *tag* nos diversos módulos, prepare um *header file*, `tagtype.h`, com a definição do tipo `MP3Tag_t` e das *macros* para dimensionar os campos.

O campo `fileName` é o ponteiro destinado a referenciar o alojamento dinâmico do espaço para o nome do ficheiro, de modo a admitir nomes com dimensão diversa.

```
typedef struct{
   char title[MAX_TIT + 1];
   char artist[MAX_ART + 1];
   char album[MAX_ALB + 1];
   short year;
   char comment[MAX_COM + 1];
   char track;
   char genre;
   char *fileName;    // Aponta string com o nome do ficheiro, em alojamento dinâmico.
} MP3Tag_t;
```

2.2. Desenvolva o módulo utilitário de acesso a ficheiro, `tag.c`, escrevendo o código e o respetivo *header file*.

Para concretizar a leitura da *tag* a partir do ficheiro, propõe-se que use as funções `fseek` e `fread` da biblioteca normalizada.

2.3. Desenvolva o módulo utilitário de gestão de dados, `setdata.c`

Considere o descritor de conjunto de *tags* com a seguinte definição:

```
typedef struct{
   int size;          // Quantidade de elementos alojados em data.
   int count;         // Quantidade de elementos preenchidos em data.
   MP3Tag_t **data;   // Indica array, alojado dinamicamente, com ponteiros para as tags.
   MP3Tag_t **aux;    // Indica array, alojado dinamicamente, com ponteiros para as tags;
} TagSet_t;           //     aux aponta as mesmas tags de data, com ordem diferente.
```

Este descritor usa, como estrutura de dados principal, apontado pelo campo `data`, um *array* de ponteiros para `MP3Tag_t`, alojado dinamicamente, o qual armazena os endereços dos descritores de *tag*, também alojados dinamicamente e preenchidos por leitura dos ficheiros MP3.

A dimensão do *array* de ponteiros deve adaptar-se automaticamente à quantidade de faixas de áudio catalogadas, com recurso à função `realloc` da biblioteca normalizada. Por razões de eficiência, quando redimensionar deve adicionar espaço para vários elementos. A gestão da dimensão e da ocupação do *array* é apoiada pelos campos `size` e `count`.

Para responder eficientemente aos comandos `a` e `t`, é conveniente dispor do acesso ordenado aos dados das *tags*, referenciados no *array*. Seria possível ordenar o *array* de ponteiros com o critério de cada um destes comandos, cada vez que fosse executado. No entanto, isso não é recomendável porque implica a espera por tempo de processamento e o desperdício de trabalho já realizado, por exemplo se executar alternadamente os dois comandos.

A estratégia proposta, para manter as duas ordenações, consiste em ordenar o *array* principal segundo um dos critérios e resolver o outro critério através de um segundo *array* de ponteiros, associado ao campo `aux`. Inicialmente prepara-se o *array* auxiliar por cópia do *array* principal, referenciando os mesmos elementos; seguidamente, ordena-se o *array* de ponteiros auxiliar com o segundo critério; a partir daí, o acesso através de cada um dos *arrays* obtém os dados pela ordem

respetiva. Por exemplo, pode ordenar o *array* principal pelo critério do comando `a` e o *array* auxiliar pelo critério do comando `t`.

A preparação do *array* auxiliar e as ordenações devem ser realizadas na função `setReady` com recurso à função `qsort` da biblioteca normalizada.

Escreva o código do módulo e o respetivo *header file*. Este deve, além das assinaturas das funções de interface, deve conter a definição do tipo `TagSet_t`.

2.4. Desenvolva o módulo utilitário para processamento da lista de ficheiros, `listfiles.c`, escrevendo o código e o respetivo *header file*.

2.5. Desenvolva o módulo de aplicação.

Prepare o *makefile* para controlar, de forma eficiente, a produção do executável;

Escreva e teste o código da aplicação, de acordo com a especificação.

É disponibilizado, no Moodle, um arquivo com amostras de faixas de áudio para teste do programa.

ISEL, 2020.05.15