Instituto Superior de Engenharia de Lisboa

LEETC – LEIRT

Programação II

2020/21 – 2.º semestre letivo

Segunda Série de Exercícios

1. Introdução

Esta série de exercícios tem como principais objetivos a organização do código em módulos e o desenvolvimento de estruturas de dados dinâmicas, nomeadamente com aplicação de *arrays* dinâmicos baseados no uso da função `realloc` da biblioteca normalizada.

Os exercícios aplicam os mecanismos estudados na SE1, envolvendo a leitura e tratamento da informação representada nas diretorias do sistema de ficheiros.

Pretende-se o desenvolvimento de um programa para armazenar os nomes dos ficheiros existentes numa diretoria e nas subdiretorias que dela dependem, admitindo em seguida, através de *standard input*, comandos para listar os nomes encontrados ou aplicar pesquisas sobre os mesmos.

2. Funcionalidades

No início, o programa deve percorrer a árvore de diretorias, usando um algoritmo recursivo e tomando como raiz a diretoria indicada por argumento de linha de comando, de modo a identificar todos os ficheiros e armazenar informação sobre eles numa estrutura de dados dinâmica. Por cada ficheiro, regista o nome, o caminho da sua localização na árvore de diretorias e a posição da sua terminação, no caso de ser uma das reconhecidas. Não se pretende registar informação específica das diretorias nem de outros itens que não sejam ficheiros.

Após a leitura das diretorias e preenchimento da estrutura de dados, o programa deve esperar, em ciclo, os comandos seguintes:

- `o` (*original*) Apresenta a lista de todos os ficheiros pela ordem em que estes foram encontrados nas diretorias percorridas. Mostra, para cada um, o caminho da sua localização, o nome completo e a indicação da terminação identificada, se existir.
- `n` (*name*) Apresenta a lista de todos os ficheiros pela ordem alfabeticamente crescente do nome completo. No caso de haver nomes iguais, a estes aplica-se a ordem alfabeticamente crescente dos caminhos da respetiva localização. O comando mostra, para cada ficheiro, o caminho da sua localização, o nome completo e a indicação da terminação identificada, se existir.
- `t` *`word`* (*termination*) Apresenta a lista parcial dos ficheiros que têm a terminação indicada pelo parâmetro *`word`*. A ordem de apresentação é idêntica à do comando "n". O comando mostra, para cada ficheiro, o caminho da sua localização e o nome completo.
- `s` *`word`* (*search*) Apresenta a lista parcial dos ficheiros que, excluindo a terminação (no caso de existir), têm o nome indicado pelo parâmetro *`word`*. A ordem de apresentação é idêntica à do comando "n". O comando mostra, para cada ficheiro, o caminho da sua localização e o nome completo.
- `q` (*quit*) Termina.

Sugere-se a criação de diretorias de ensaio com diversidade de conteúdo para o teste e demonstração do programa.

3. Estruturas de dados

3.1. Coleção de *strings* partilhadas

Na estrutura que representa a informação dos ficheiros, com a forma especificada adiante, ocorre o uso de *strings* idênticas em vários elementos, para representar o caminho da localização dos ficheiros na mesma diretoria. Com vista a evitar o desperdício de memória com réplicas da mesma *string*, pretende-se criar uma estrutura de dados para armazenar conjuntos de *strings* e disponibilizar o acesso partilhado às mesmas.

A estrutura de dados para as *strings* partilhadas deve manter registo das mesmas, de modo a permitir a posterior libertação da memória de alojamento dinâmico que elas ocupam. Propõe-se a criação do tipo `StrShare` para representar o descritor de um conjunto de *strings* partilhadas.

```
typedef struct {
    int space; // quantidade de elementos alojados para o array data
    int count; // quantidade de elementos preenchidos
    char **data; // array de ponteiros para as strings existentes
} StrShare;
```

As *strings* armazenadas no conjunto são alojadas dinamicamente e referenciadas pelos elementos do *array* indicado pelo campo `data` do descritor. Este *array* também é alojado dinamicamente, sendo redimensionado com recurso à função `realloc`. O redimensionamento deve ser realizado em blocos de vários elementos, por motivos de eficiência.

3.2. Descritor de ficheiro

Propõe-se o tipo `FileInfo`, para representar a informação associada ao nome de um ficheiro.

```
typedef struct {
    char *path; // string partilhada – localização do ficheiro
    char *name; // string alojada dinamicamente – nome completo
    int term;   // posição da terminação no nome
} FileInfo;
```

Em cada elemento do tipo `FileInfo`, o campo `path` deve ser apontado para uma *string* (caminho) armazenada num conjunto de *strings* partilhadas, com registo numa estrutura do tipo `StrShare`; o campo `name` aponta uma *string* também alojada dinamicamente, mas não partilhada. O campo `term` deve ser afetado de acordo com a especificação do campo `term` no tipo `Item` da SE1.

3.3. *Arrays* de referências a ficheiros

Propõe-se o tipo `RefArray`, como descritor de um *array* dinâmico que contém referências a elementos do tipo `FileInfo`.

```
typedef struct {
    int space; // quantidade de elementos alojados para o array data
    int count; // quantidade de elementos preenchidos
    FileInfo **data; // array de ponteiros
} RefArray;
```

As referências são armazenadas nos elementos do *array* indicado pelo campo `data` do descritor. Este *array* é alojado dinamicamente, sendo redimensionado com recurso à função `realloc`. O redimensionamento deve ser realizado em blocos de vários elementos, por motivos de eficiência.

Propõe-se a criação de dois *arrays* dinâmicos com o tipo `RefArray`, um que referencia o conjunto dos descritores de ficheiro na ordem original, para responder ao comando "o", e outro com a ordem conveniente para os comandos "n", "t" e "s". A construção da estrutura de dados consiste em criar descritores do tipo `FileInfo`, adicionar as respetivas referências aos dois *arrays* citados e, no final, ordenar o *array* de referencias destinado a responder aos comandos "n", "t" e "s".

4. Organização em módulos

Organize o desenvolvimento do programa prevendo, pelo menos os seguintes módulos:

- Gestão de *strings* partilhadas;
- Identificação da terminação no nome de um ficheiro, com a funcionalidade especificada na SE1;
- Manipulação de descritores de ficheiro;
- Armazenamento e manipulação de *arrays* de referências para descritores de ficheiro;
- Percurso na árvore de diretorias para identificar os nomes e localizações dos ficheiros;
- Aplicação que contém, pelo menos, a função `main`.

Por cada módulo fonte (.c) deve escrever um *header file* respetivo (.h) com a definição dos tipos necessários e as assinaturas das funções de interface.

Deve construir um *makefile* para controlar a produção e atualização do executável. É conveniente testar o código ao longo do desenvolvimento, realizando a aplicação por fases ou criando aplicações de teste parcial.

4.1. Escreva o módulo de gestão de *strings* partilhadas de modo a dispor das funções de interface seguintes.

```
StrShare *strShareCreate( void );
```

Aloja dinamicamente o descritor para um conjunto de *strings*, inicia-o no estado vazio e retorna o seu endereço. O campo `data`, destinado a referenciar um *array* de ponteiros para as *strings*, deve ser iniciado de acordo com a posterior utilização da função `realloc`.

```
void strShareDelete( StrShare *share );
```

Liberta todo o espaço de alojamento dinâmico sob controlo do descritor indicado por `share` , incluindo as *strings* nele registadas.

```
char *strShareAdd( StrShare *share, char *str );
```

Adiciona ao conjunto identificado por `share` uma réplica da *string* indicada por `str` e retorna o seu endereço.

4.2. Escreva o módulo de identificação da terminação, adaptando a função `findTerm` desenvolvida na SE1.

```
void termSetupTypes( char *terms[], int numTerms );
```

Recebe parâmetros que identificam os tipos de terminação considerados e regista-os, em variáveis internas do módulo, para futura utilização na função `termFind` seguinte. O parâmetro `terms` indica um *array* de ponteiros para *strings* com as terminações a considerar; O parâmetro `numTerms` representa o número de terminações identificadas pelos elementos de `terms`. Admite-se que o conteúdo do *array* passado neste parâmetro permanece sem alterações até ao fim da atividade do programa.

```
int termFind ( char *str );
```

Verifica se o nome de ficheiro indicado por `str` tem uma das terminações consideradas, previamente registadas em variáveis do módulo pela função `termSetupTypes`. A função `termFind` retorna o índice, na *string* `str`, do '.' que antecede a terminação. Se não existir nenhuma das terminações previstas, retorna o número total de carateres da *string* indicada por `str`.

4.3. Escreva o módulo de manipulação de descritores de ficheiro, com as funções de interface seguintes.

```
FileInfo *fileInfoNew( char *sharedPath, char *name );
```

Cria um novo descritor de ficheiro, com o caminho e nome indicados por `sharedPath` e `name`. O endereço `sharedPath` deve ser simplesmente registado no descritor, dado que indica uma *string* já alojada para uso partilhado. A *string* indicada por `name` deve ser copiada para espaço alojado dinamicamente para uso específico do descritor agora criado. Deve utilizar a função `termFind` para determinar o valor do campo `term`.

```
void fileInfoDelete( FileInfo *info );
```

Elimina o descritor de ficheiro indicado por `info`, libertando todo o espaço de alojamento dinâmico que está na sua posse.

4.4. Escreva o módulo de armazenamento e manipulação de referências, com as funções de interface seguintes.

```
RefArray *refArrCreate( void );
```

Aloja dinamicamente o descritor para um *array* dinâmico de referências para descritores de ficheiro, inicia-o no estado vazio e retorna o seu endereço. O campo `data`, destinado a referenciar um *array* de ponteiros para os elementos `FileInfo`, deve ser iniciado de acordo com a posterior utilização da função `realloc`.

```
void refArrDelete( RefArray *ra);
```

Liberta o espaço de alojamento dinâmico ocupado pelo descritor indicado por `ra` e pelo *array* de ponteiros que ele controla.

```
void refArrAdd( RefArray *ra, FileInfo *ref );
```

Adiciona ao *array* dinâmico identificado por `ra` a referência `ref`.

```
void refArrSort( RefArray *ra );
```

Ordena as referências armazenadas no *array* dinâmico identificado por `ra`, aplicando a ordem conveniente para os comandos "n", "t" e "s" da aplicação. Deve utilizar a função `qsort` da biblioteca normalizada.

```
void refArrScan( RefArray *ra, void (*act)(FileInfo *fi, void *param),
                 void *actParam );
```

Percorre o *array* identificado por `ra` aplicando, a cada elemento referenciado, a função passada no parâmetro `act`. Quando executar a função indicada por `act` deve passar-lhe, no respetivo parâmetro `param`, o ponteiro recebido em `actParam`.

A função `refArrScan` destina-se a executar ações repetitivas como, por exemplo, a exibição dos elementos referenciados ou a sua eliminação. O parâmetro `actParam` permite passar dados adicionais; por exemplo na implementação dos comandos "t" e "s", permite passar uma *string* às funções passadas em `act` para escolher os elementos a apresentar.

4.5. Escreva o módulo de leitura da árvore de diretorias, com a função de interface seguinte.

```
void scanDirTree( char *path, StrShare *pathShare,
                  RefArray *origRef, RefArray *sortRef );
```

Percorre a árvore de diretorias, a partir da indicada em `path`, cria os descritores de ficheiro e referencia-os nos conjuntos indicados por `origRef` e `sortRef`.

O descritor indicado por `pathShare` serve para armazenar as *strings*, com caminhos, que irão ser partilhadas pelos descritores de vários ficheiros.

O conjunto `origRef` destina-se a manter a informação dos ficheiros na ordem original. O conjunto `sortRef`, após o preenchimento, é ordenado para responder aos comandos "n", "t" e "s" da aplicação.

Para ler os dados da diretoria propõe-se a utilização das funções, de biblioteca normalizada, `opendir`, `readdir` e `closedir`. O percurso na árvore de diretorias deve ser realizado com um algoritmo recursivo, em que o caminho de acesso a cada subdiretoria é obtido por concatenação do seu nome ao caminho da diretoria anterior.

4.6. Escreva o módulo aplicação com a implementação da função `main` e de outras que sejam convenientes. São da responsabilidade da aplicação as seguintes atividades:

- Interpretar os parâmetros de linha de comando;
- Criar e preencher os descritores para armazenar informação, recorrendo aos outros módulos;
- Interpretar os comandos do utilizador e acionar a sua resposta, recorrendo aos outros módulos;
- Na fase de terminação, eliminar as estruturas de dados dinâmicas e libertar a memória que elas ocupam.

4.7. Realize o teste completo do código realizado. Prepare e ensaie um conjunto de cenários demostrativos, com diversidade de ficheiros, incluindo nomes diferentes, mas também nome iguais em subdiretorias diferentes.