Instituto Superior de Engenharia de Lisboa

LEETC

Programação II

2024/25 – 1.º semestre letivo

Segunda Série de Exercícios

Esta série de exercícios incide principalmente sobre os conceitos de função genérica, utilizando funções passadas por parâmetro, nas vertentes de desenvolvimento e de utilização.

Os exercícios envolvem o armazenamento de dados em *arrays* e a respetiva ordenação. Especifica-se a declaração desses *arrays* com dimensão fixa. Não se pretende o uso de alojamento dinâmico nesta série de exercícios.

O desenvolvimento deve ser realizado em módulos separados; propõe-se que agrupe as funções por famílias com alguma relação entre si; um módulo poderá conter uma função ou um grupo de funções relacionadas, sendo uma ou mais públicas. No caso de haver funções que sejam úteis apenas no respetivo módulo, devem ser definidas com *scope* privado.

Os alunos devem escrever os módulos fonte com as funções especificadas nos exercícios, podendo adicionar outras que considerem convenientes, e definir os *header files* (.h) adequados para partilhar os tipos de dados e as assinaturas das funções públicas, dispondo de controlo da inclusão múltipla. Prevê-se a reutilização de código anteriormente desenvolvido, aproveitando os módulos anteriormente escritos, ou reorganizando as funções em novos módulos, acompanhados dos respetivos *header files*. Pretende-se também a preparação de ficheiros *makefile* para gerar, de forma eficiente, os executáveis dos programas especificados.

1. Leitura de um ficheiro de texto, com processamento linha a linha, parametrizável

Pretende-se o desenvolvimento de uma função capaz de aplicar um processamento, parametrizável, a todas as linhas de um ficheiro de texto codificado em ASCII simples.

1.1. Escreva a função

```
int processFile( const char *filename,
                 int (*action)( const char *line, void *context ),
                 void *context );
```

que abre o ficheiro com o nome indicado, lê o seu conteúdo linha a linha e aplica a cada linha o processamento implementado pela função passada em `action`. Ao executar esta função `action`, deve passar uma *string* com a linha a processar e o parâmetro `context` recebido.
A função retorna a soma dos valores retornados por todas as chamadas à função `action`.
Sugere-se que realize a leitura do ficheiro de entrada com a função `fgets`. Propõe-se a dimensão de 512 carateres para a memória destinada ao armazenamento da linha.

1.2. Escreva a função

```
int linePrintRaw( const char *line, void *context );
```

destinada a ser passada no parâmetro `action` da função anterior para apresentar, em *standard output*, o conteúdo integral da linha de texto indicada em `line`.
A função retorna 1. O parâmetro `context` não é usado neste exercício; existe para compatibilidade de assinatura, mas deve ser ignorado pelo código da função.

1.3. Escreva e teste um programa (com o nome `prog13.c` e executável `prog13`) para demostração das funções anteriores. O programa recebe, em argumento de linha de comando, o nome de um ficheiro e apresenta, em *standard output*, o seu conteúdo integral.

1.4. Pretende-se usar a função `processFile` para filtrar ficheiros com linhas organizadas em campos, mostrando apenas as linhas que têm um determinado conteúdo no primeiro campo.
Escreva a função

```
int lineFilterPrint( const char *line, void *context );
```

destinada a ser passada no parâmetro `action` da função `processFile` para apresentar, em *standard output*, o conteúdo integral da linha indicada por *line*, se o seu primeiro campo for considerado idêntico à *string* indicada por `context`. Deve utilizar as funções da SE1, `splitField` para separar o primeiro campo, `separatorUnify` para o uniformizar e `strcmp_ic` para a comparação. A função retorna 1, se a linha é apresentada, ou 0, no caso contrário.

1.5. Escreva e teste um programa (com o nome `prog15.c` e executável `prog15`) para demostração da função anterior. O programa recebe, nos argumentos de linha de comando, o nome de um ficheiro e uma *string*; apresenta, em *standard output*, o conteúdo integral das linhas cujo primeiro campo for considerado idêntico à *string* passada no comando, após unificada pela função `separatorUnify`. No final, o programa deve mostrar o número de linhas apresentadas.

2. Armazenamento, ordenação e pesquisa de uma lista de livros

Considere a informação relativa a uma lista de livros armazenada num ficheiro de texto como o exemplo anexo, cujas linhas contêm a seguinte sequência de dados: Título; ISBN; ISBN-13; Autor(es); Editor; Palavras chave; Número de páginas; Data de publicação; Tipo de encadernação; Preço.

Propõe-se a criação dos tipos `BookData`, como elemento de uma estrutura de dados para representar os dados dos livros, e `Collection` como descritor de um conjunto de dados bibliográficos.

```
typedef struct book {
    char title[MAX_TITLE];
    char isbn[SIZE_ISBN];
    char authors[MAX_AUTHORS];
    char publisher[MAX_PUB_NAME];
} BookData;

typedef struct{
    BookData books[MAX_BOOKS];
    BookData *refs[MAX_BOOKS];
    int count; // quantidade de elementos preenchidos em books
} Collection;
```

Pretende-se o desenvolvimento de um programa para armazenar a lista de livros, disponibilizando comandos para apresentar o seu conteúdo ou parte dele. O programa deve usar a função `processFile` para ler e armazenar os dados, a partir de um ficheiro indicado por argumento de linha de comando, armazenando-a no *array* `books` de um descritor do tipo `Collection`. Os dados são apresentados em resposta a comandos introduzidos através de *standard input*.

Considerando a criação das variáveis sem recurso a alojamento dinâmico, admita um cenário simplificado em que existem dimensões máximas para os elementos de informação – livros, carateres do título e carateres dos nomes dos autores ou do editor. Estas dimensões podem ser identificadas a partir do ficheiro de dados, com a ajuda dos comandos do Linux, por exemplo: «`wc -l -L` *file*» para obter, relativamente a um ficheiro, a quantidade de linhas e a dimensão da mais longa; «`cat` *file* `| cut -d";" -f` *field_number* `| wc -L`» para identificar a maior dimensão de um determinado campo. Recomenda-se o sobredimensionamento das quantidades observadas.

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

- `l` Apresenta a lista de todos os livros, com ordem alfabeticamente crescente por título, indicando, para cada um: título, autor, editor e ISBN;
- `a` *name* Apresenta um subconjunto da lista de livros, com ordem alfabeticamente crescente por título, indicando, para cada um: título, autor, editor e ISBN; Os livros selecionados são os que contêm a palavra *name* no campo de autores. A verificação da palavra deve ser insensível a maiúsculas ou minúsculas;
- `i` *isbn* Apresenta os dados relativos ao livro com o ISBN indicado; Se este não existir, apresenta um aviso;
- `q` Termina.

2.1. Identifique, usando os comandos propostos, as dimensões para a estrutura de dados e defina as respetivas macros.

2.2. Escreva a função

```
int fillBookData( BookData *b, const char *line );
```

destinada a armazenar a informação de um livro. Deve identificar os campos da linha indicada por `line` e copiar os relevantes para a estrutura indicada por `b`, uniformizando os respetivos conteúdos. Deve usar as funções `splitField` e `separatorUnify` da série anterior.
A função `fillBookData` retorna: 1, em caso de sucesso; 0, se a linha não tiver o conteúdo adequado.

2.3. Escreva a função

```
int collAddBook( const char *line, void *context );
```

destinada a ser passada no parâmetro `action` da função `processFile` para adicionar os dados de um livro ao armazenamento de uma coleção.
A linha de texto indicada por `line` contém os campos de informação, na forma obtida do ficheiro. O parâmetro `context` representa o endereço do descritor do tipo `Collection` que armazena os dados. Nesta, o campo *array* `books` destina-se a armazenar os dados, com um elemento por livro, e o campo `count` representa a quantidade de elementos preenchidos. Neste exercício não é usado o campo *array* `refs`.
Tendo em conta a capacidade limitada do descritor, se a quantidade de livros for excessiva, são considerados os dados na parte inicial do ficheiro. Deve utilizar a função `fillBookData`, para adicionar o preenchimento do novo elemento.
A função retorna 1, em caso de sucesso, ou 0, no caso contrário, devido a linha incorreta ou a capacidade insuficiente do descritor.

2.4. Escreva e teste um programa (com o nome `prog24.c` e executável `prog24`) para demostração das funções anteriores. O programa recebe, nos argumentos de linha de comando, o nome do ficheiro de dados; usa um descritor do tipo `Collection`, que preenche com os dados obtidos; no final apresenta a totalidade dos dados armazenados, pela ordem em que se encontram no ficheiro, reproduzindo os campos em *standard output*, novamente separados por ponto-e-vírgula.

2.5. Com o propósito de permitir a listagem ordenada por título, para os comandos “l” e “a”, escreva a função

```
void collSortTitle( Collection *col );
```

que ordena por título os livros existentes no campo *array* `books` da coleção indicada por `col`. A ordem é alfabeticamente crescente, insensível a maiúsculas ou minúsculas.
Deve utilizar a função `qsort` da biblioteca normalizada.

2.6. Escreva e teste o programa parcial de aplicação (com o nome `prog26.c` e executável `prog26`) capaz de aceder ao ficheiro, preencher os dados e responder aos comandos "l" e "q" especificados.

2.7. Com vista à implementação do comando "i", tendo como objetivo a eficiência, é necessário criar um acesso aos livros ordenado por ISBN. Contudo, pretende-se manter inalterada a ordenação já realizada do campo *array* `books`, para voltar a responder ao comando "l". Assim, usa-se outro *array*, este de ponteiros (campo `refs`), os quais serão ordenados para suportar o acesso aos dados com um critério de ordenação diferente.
Escreva a função

```
void collSortRefIsbn( Collection *col );
```

destinada a ser executada após o preenchimento e ordenação do *array* `books`. Esta função deve iniciar o campo *array* de ponteiros `refs`, apontando para as respetivas posições em `books`, de modo a referenciar os elementos de dados; o *array* de ponteiros deve então ser ordenado de modo a disponibilizar o acesso aos dados por ordem de ISBN. Para ordenar, deve utilizar de novo a função `qsort`.

2.8. Escreva e teste o programa parcial de aplicação (com o nome `prog28.c` e executável `prog28`) adicionando o código necessário para a implementação do comando "i". Na resposta a este comando, a pesquisa deve ser realizada com a função `bsearch` da biblioteca normalizada.

2.9. Como propósito de suportar o comando "a", escreva a função

```
int bookContainsAuthor( BookData *b, const char *word );
```

que verifica se a palavra indicada por `word` existe, como palavra isolada, no campo de autores do livro indicado por `b`. Em caso afirmativo, retorna 1; se não, retorna 0.
Propõe-se que utilize a função `strtok`, da biblioteca normalizada, sobre uma cópia dos nomes de autores do livro, de modo a isolar as respetivas palavras, e a função `strcmp_ic` da série anterior para comparar com a palavra pesquisada.

2.10. Escreva e teste o programa completo de aplicação (com o nome `prog210.c` e executável `prog210`) adicionando o código necessário para a implementação do comando "a". Na resposta a este comando, deve percorrer os dados de acordo com a ordenação pretendida e utilizar a função `bookContainsAuthor` para selecionar os livros a apresentar.

ISEL, 2024.10.21