Instituto Superior de Engenharia de Lisboa

LEETC

Programação II

2022/23 – 1.º semestre letivo

Segunda Série de Exercícios

Esta série exercita os conceitos de função genérica, utilizando funções passadas por parâmetro, tanto na vertente de desenvolvimento como de utilização. São também exploradas funcionalidades da biblioteca normalizada para acesso a ficheiros.

No desenvolvimento destes exercícios não é exigida a programação em módulos separados. No entanto, os alunos que já tiverem domínio desta metodologia poderão utilizá-la; neste caso, devem definir os *header files* (.h) adequados para partilhar os tipos de dados e as assinaturas das funções públicas.

1. Separação e processamento de palavras existentes numa *string*.

   1.1. Escreva a função

   ```
   int wordSplit( char *str, int (*proc)( char *word, void *context ),
                  void *context );
   ```

   que separa as palavras existentes na *string* `str`, usando a função `strtok` da biblioteca normalizada, e invoca sucessivamente a função passada em `proc`, passando o endereço de cada uma das palavras identificadas e o parâmetro `context`. Retorna a soma dos valores retornados pelas sucessivas chamadas à função passada em `proc`.

   1.2. Escreva a função

   ```
   int wordPrint( char *word, void *context );
   ```

   destinada a ser passada no parâmetro `proc` da função anterior para apresentar, numa *stream* de saída, a palavra passada no parâmetro `word`. O parâmetro `context` representa a *stream* onde são exibidas as palavras. A função retorna sempre 1.

   1.3. Escreva e teste um programa de demostração que deve receber uma sequência de linhas de texto, através de *standard input* com a função `fgets`, e invocar a função `wordSplit`, passando em `str` a linha de texto, em `proc` a função `wordPrint` e em `context` a *stream* `stdout`, de modo a apresentar em *standard output* as palavras existentes em cada linha. Utiliza o valor de retorno de `wordSplit` para apresentar a quantidade de palavras de cada linha, após o processamento da mesma.

2. Pretende-se um programa de aplicação, designado por `wsearch` (*word search*) para pesquisar palavras em ficheiros de texto. Os textos, bem como as palavras a pesquisar, são codificados em UTF-8; a verificação das palavras deve ser insensível ao uso de maiúsculas ou minúsculas, tanto nos textos como nas palavras indicadas pelo utilizador.

   O programa deve receber, nos argumentos de linha de comando, os nomes dos ficheiros de texto para pesquisa. Note que se usar *wildcards* nos argumentos da execução, o interpretador de comandos expande essa forma para os nomes de ficheiro que satisfaçam o padrão e passa-os ao executável. Por exemplo, se existirem os ficheiros "`a.txt`", "`b.txt`" e "`c.txt`", o comando "`wsearch *.txt`" é equivalente a "`wsearch a.txt b.txt c.txt`".

Na interação com o utilizador, recebe as palavras a pesquisar e procura-as sequencialmente nos ficheiros. Sempre que encontrar a palavra pesquisada, deve apresentar, através de *standard output*:

- O nome do ficheiro;
- O número da linha que contém a palavra;
- O texto original dessa linha (sem o efeito da normalização).

Na construção deste programa, os alunos devem reutilizar algumas funções desenvolvidas anteriormente, na série de exercícios anterior e na atual, e desenvolver a componente de aplicação.

2.1. Organize a integração neste exercício das funções `wordSplit`, `normString` e outras de que esta dependa.

2.2. Escreva a função
`int wordCheck( char *word, void *context );`
adequada para ser passada no parâmetro `proc` da função `wordSplit` com o propósito de verificar se cada palavra passada no parâmetro `word` é a que se pretende pesquisar.
O parâmetro `context`, neste caso, é usado como ponteiro para a *string* com a palavra a pesquisar. Tendo em conta a existência de carateres acentuados, deve utilizar a função `strcoll`, de biblioteca, para realizar as comparações (ver anexo, no final, sobre a função `strcoll`).
A função `wordCheck` retorna 1, se a palavra é a pesquisada, ou 0, no caso contrário.

2.3. Escreva e teste o código da aplicação.
Para a orgânica do programa, propõe-se a estratégia seguinte:

- Permanece em ciclo, recebendo do utilizador as palavras a pesquisar;
- Após receber e normalizar cada palavra a pesquisar, percorre a lista de nomes dos ficheiros, passados nos parâmetros da linha de comando;
- Abre cada ficheiro, lê sequencialmente as linhas de texto, processa-as e fecha o ficheiro;
- O processamento de cada linha consiste em preparar uma cópia normalizada da linha e realizar, sobre ela, a separação das palavras e a respetiva verificação da pesquisa, usando a função `wordSplit` com `wordCheck`.
- Quando, ao processar uma linha, a função `wordSplit` retorna um valor diferente de 0, isso significa que contém a palavra pesquisada, pelo que a aplicação deve apresentar os elementos especificados;
- A atividade termina quando o utilizador acionar o fim de inserção (premindo Ctrl-D, equivalente a *end-of-file*).

3. Pretende-se o armazenamento e ordenação das palavras, com vista ao desenvolvimento futuro de um método de pesquisa mais eficiente, evitando a leitura sequencial dos ficheiros por cada palavra procurada. Este exercício inclui um programa de demonstração que começa por realizar o armazenamento das palavras; em seguida apresenta uma lista geral das mesmas; por último aceita palavras para demostração da pesquisa e responde a cada palavra indicando se esta existe ou não.

   O armazenamento das palavras é sem repetições, isto é, regista-se somente um exemplar de cada palavra existente nos textos. Para armazenar as palavras propõe-se a criação de um *array* bidimensional de `char`; as dimensões (máximo de carateres por palavra e quantidade máxima de palavras) são fixas, com o propósito de simplificar o exercício.

   O *array* deve ser integrado numa estrutura de dados passada ao processamento das palavras através do parâmetro `context` da função `wordSplit`.

   Tal como no exercício anterior, deve reutilizar algumas funções desenvolvidas anteriormente, e desenvolver o programa de aplicação.

3.1. Organize a integração neste exercício das funções `wordSplit`, `normString` e outras de que esta dependa.

3.2. Prepare, para utilização neste exercício, o tipo `DataStore` seguinte, destinado a armazenar as palavras.

```
typedef struct{
    int count;
    char words[MAX_WORD_COUNT][MAX_WORD_SIZE];
} DataStore;
```

3.3. Escreva a função

```
int wordStore( char *word, void *context );
```

destinada a ser passada no parâmetro `proc` da função `wordSplit` para adicionar a palavra passada no parâmetro `word` ao *array* de armazenamento. O parâmetro `context`, neste caso, é usado como ponteiro para uma estrutura do tipo `DataStore` para armazenar as palavras. Se exceder a capacidade do *array*, deve apresentar uma mensagem de erro através de `stderr`.

A função `wordStore` retorna 1, se a palavra foi adicionada, ou 0, no caso de já existir ou exceder a capacidade.

3.4. Escreva as funções seguintes:

```
int dataComp( const void *, const void * );
void dataSort( DataStore *data );
char *dataSearch( DataStore *data, const char *sample );
```

A função `dataComp` destina-se a ser passada nas funções de biblioteca `qsort` e `bsearch`, como função de comparação para avaliar a ordenação alfabética crescente de duas palavras representadas em elementos do *array* `words`, pertencente ao tipo `dataStore`. Tendo em conta a existência de carateres acentuados, deve utilizar a função `strcoll`, de biblioteca, para realizar as comparações.

A função `dataSort` deve chamar `qsort` com `dataComp` para realizar a ordenação dos elementos preenchidos na estrutura indicada por `data`.

A função `dataSearch` deve chamar `bsearch` com `dataComp` para realizar a pesquisa da palavra `sample` nos elementos preenchidos na estrutura indicada por `data`. Retorna o ponteiro para a palavra encontrada ou NULL, se não existir.

3.5. Escreva e teste o programa de demonstração.

Para a orgânica do programa, propõe-se a estratégia seguinte:

- Define a estrutura de dados para armazenamento das palavras e inicia-a no estado vazio;
- Percorre a lista de nomes dos ficheiros, passados nos parâmetros da linha de comando;
- Abre cada ficheiro, lê sequencialmente as linhas de texto, processa-as e fecha o ficheiro;
- No processamento de cada linha, realiza a sua normalização, a separação das palavras e a respetiva adição ao *array* de armazenamento, usando a função `wordSplit` com `wordStore`;
- Ordena as palavras, utilizando a função `dataSort`, e apresenta a lista ordenada;
- Permanece em ciclo, recebendo do utilizador as palavras a verificar;
- Após receber e normalizar cada palavra, verifica se ela existe no *array*, utilizando `dataSearch`.
- A atividade termina quando o utilizador acionar o fim de inserção.

A. **Anexo** – Textos para ensaiar os exercícios

Com o propósito de promover o ensaio do programa com muitas palavras, de modo a evidenciar o tempo de processamento recomenda-se a utilização de textos com dimensão significativa. Propõe-se, por exemplo, o acesso ao repositório http://figaro.fis.uc.pt/queiros/lista_obras.html que disponibiliza o texto integral de diversas obras do escritor Eça de Queirós.

Propõe-se, ainda, que os alunos preparem excertos dos textos referidos ou ficheiros escritos propositadamente para testar os exercícios em cenários de ensaio específicos.

B. **Anexo** – Ordenação de *strings* com carateres acentuados

Para comparar palavras por ordem alfabética, respeitando a acentuação, a função `strcmp` não produz a ordem mais adequada, devido à distância entre os códigos numéricos usados na codificação das letras com acentos diversos. Propõe-se o uso da função `strcoll` de biblioteca, que está preparada para reconhecer a ordem correta, atribuindo ordem adjacente às variantes de cada letra com os diversos acentos. Esta função necessita de definições relacionadas com a língua, selecionadas pela função `setlocale`.

De modo a usar a função `strcoll` é necessário assegurar as seguintes condições:

- Dispor do *locale* para a língua pretendida. Pode verificar as variantes existentes com o comando "`locale -a`" na linha de comando.

  O *locale* "`pt_PT.UTF-8`" representa as definições específicas de Portugal. Se não estiver instalado é necessário gerá-lo, executando o comando "`sudo locale-gen pt_PT.UTF-8`".

- No programa, incluir o *header file* "`locale.h`".

- No início da atividade, executar "`setlocale(LC_ALL,"pt_PT.UTF-8");`".

- Nas comparações, usar "`strcoll( string1, string 2 );`"

ISEL, 2022.11.01