Verificador ortográfico

Este projeto foi aprimorado e atualizado para o formato UTF-8.

Este projeto visa implementar um verificador ortográfico simples, que verifica se as palavras de um texto de entrada estão em um dicionário predefinido e sugere correções para as palavras não encontradas.

Descrição

O verificador de ortografia recebe um texto de entrada e verifica se suas palavras estão em um dicionário. A saída deve **reproduzir fielmente o texto de entrada** (inclusive espaços e quebras de linha), mas colocando entre colchetes as palavras não encontradas no dicionário.

Por exemplo, para esta entrada:

```
   Para que o pocessador possa interromper a execução de uma
tarefa e retornar a ela mais tarde, sem corromper seu estado
interno, é necessário definir operações para salvar e
restaurrar o contexto da tarefa.
===
   O ato de salvar os valores do contexto atual em seu TCB e
possivelmente restaurar o contexto de outra tarefa, previamente
salvo em outro TCB, é denominado "troca de contexto".
```

O programa deve gerar esta saída:

```
   Para que o [pocessador] possa interromper a execução de uma
tarefa e retornar a ela mais tarde, sem corromper seu estado
interno, é necessário definir operações para salvar e
[restaurrar] o contexto da tarefa.
===
   O ato de salvar os valores do contexto atual em seu [TCB] e
possivelmente restaurar o contexto de outra tarefa, previamente
salvo em outro [TCB], é denominado "troca de contexto".
```

Consideram-se como palavras as sequências contíguas de letras (A-Z, a-z) com ou sem acentos e as cedilhas; os demais caracteres (números, espaços e outros símbolos) não fazem parte de palavras.

Exemplos de arquivos de texto com erros de ortografia, no formato UTF-8 (baixe com “salvar como”, para evitar conversões de caracteres indesejadas):

- plutao.txt, 2.283 bytes
- memoria.txt, 3.203 bytes
- brascubas.txt, 360.681 bytes
- montesquieu.txt, 1.288.323 bytes
- brazilian.gz, dicionário com ~275 mil palavras

Os arquivos acima e o dicionário estão no formato UTF-8, que é o padrão da maioria dos sistemas atuais. Esse formato pode usar mais de um byte por caractere para representar letras acentuadas e sinais gráficos. Se for usar outros textos, assegure-se de que também estejam no formato UTF-8.

Sugestões

Um bom verificador ortográfico deve **sugerir correções** para as palavras incorretas. Programe o verificador para informar as palavras conhecidas mais próximas das palavras não encontradas no dicionário, como mostra este exemplo:

```
   Para que o [pocessador (processador)] possa interromper a execução de uma
tarefa e retornar a ela mais tarde, sem corromper seu estado
interno, é necessário definir operações para salvar e
[restaurrar (restaurar)] o contexto da tarefa.
===
   O ato de salvar os valores do contexto atual em seu [TCB (aba)] e
possivelmente restaurar o contexto de outra tarefa, previamente
salvo em outro [TCB (aba)], é denominado "troca de contexto".
```

Para encontrar a palavra mais próxima no dicionário, sugere-se usar a distância Levenshtein, que permite calcular a distância entre duas strings.

O algoritmo de Levenshtein é lento, portanto evite comparar palavras de comprimentos muito diferentes (por exemplo, evite calcular a distância entre “uva” e “laranja”).

Forma de chamada

```
$ ortografia [-s] [ -d dicionário ] [ -i entrada ] [ -o saída ]
```

Opções:

- `-i` : indica o arquivo de entrada; se não for informado, assume-se a entrada padrão (*stdin*).
- `-o` : indica o arquivo de saída; se não for informado, assume-se a saída padrão (*stdout*).
- `-d` : indica o arquivo de dicionário, em formato UTF-8; se não for informado, assume-se o dicionário do sistema operacional em `/usr/share/dict/brazilian`.
- `-s` : ativa a sugestão de correções usando a distância de Levenshtein (por default desativada).
- `-h` : gera uma mensagem de ajuda na saída de erro (*stderr*), explicando o que o programa faz e quais as opções disponíveis.
- Todas as mensagens de erro devem ser enviadas para a saída de erro (*stderr*).

Essas opções podem ser usadas em qualquer ordem:

```
// entrada e saída em arquivos
ortografia -i input.txt  -o output.txt
ortografia -o output.txt -i input.txt

// entrada em arquivo, saída em stdout, vice-versa ou ambos
ortografia -i input.txt  > output.txt
```

```
ortografia -o output.txt < input.txt
ortografia <  input.txt  > output.txt

// as opções podem estar em qualquer ordem
ortografia -d brazilian -i input.txt -o output.txt
ortografia -i input.txt -d brazilian -o output.txt -s
ortografia -o output.txt -s -i input.txt -d brazilian
```

Atividade

A implementação deve atender os seguintes requisitos:

- Funcionar corretamente com os exemplos desta página 
- O dicionário deve ser totalmente carregado em um vetor de palavras na memória RAM antes de ser usado, usando alocação dinâmica (os mais ousados podem tentar implementar este projeto usando uma árvore de prefixos).
- A localização das palavras no dicionário deve usar um algoritmo de busca binária.
- O executável deve se chamar `ortografia`.
- Usar `Makefile` com ao menos os alvos `all`, `clean` e `purge`.
- Não gerar *warnings* ao compilar com a opção `-Wall`.
- Processar o arquivo `brascubas.txt` em **menos de 1 segundo** (com sugestões desativadas).

Para medir o tempo de execução do programa, use o comando `time`:

```
time ./ortografia -i brascubas.txt -o output.txt
```

Arquivos a entregar ao professor:

- `ortografia.c` : programa principal
- `dicionario.c` : funções relativas ao dicionário (carregar o dicionário na memória, verificar se uma palavra está no dicionário, etc);
- `dicionario.h` : interface (protótipos) das funções implementadas em `dicionario.c`;
- `Makefile`

Dica: as funções da biblioteca C padrão (StdLib) podem facilitar a implementação de seu programa:

- Acesso a arquivos: `fopen`, `fclose`, `fgetchar` `fscanf`, `feof`, ...
- Uso de caracteres e strings largas: `getwchar`, `putwchar`, `iswalpha`, `towlower`, `wcslen`, `fwscanf`, `wcscpy`, `wcscasecmp`, ...
- Busca binária: `bsearch`
- Ordenação: `qsort`

Consulte as páginas de manual para aprender a usar essas funções.

Sugestões de implementação

Por pseudocódigo:

```
ler o dicionário em um vetor de palavras

c = ler caractere da entrada
```

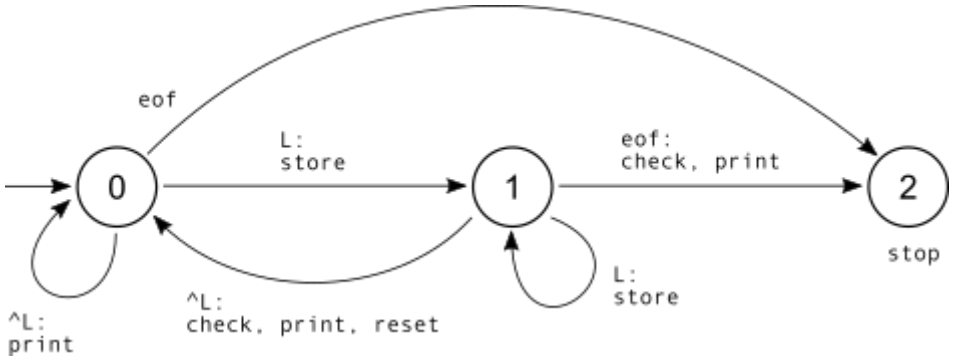
```
enquanto não for o fim da entrada faça
  // avançar até encontrar uma letra ou o fim da entrada
  enquanto (c não for uma letra) e (não for o fim da entrada) faça
    escrever c em stdout
    c = ler caractere da entrada
  fim enquanto

  // encontrou uma letra, ler a palavra inteira
  palavra = ""
  enquanto (c for uma letra) e (não for o fim da entrada) faça
    palavra = palavra + c
    c = ler caractere da entrada
  fim enquanto

  // tratar a palavra encontrada
  se palavra <> "" então
    se minúscula(palavra) está no dicionário então
      escrever palavra na saída
    senão
      escrever "[", palavra, "]" na saída
    fim se
  fim se

fim enquanto
```

Por máquina de estados:




```
state = 0
while state ≠ 2 do

   read input

   case (state)

      0: case (input)

            letter:
               append input to word
               state = 1

            not letter:
               print input

            eof:
               state = 2
```

```
         end case

      1: case (input)

            letter:
               append input to word

            not letter:
               check word
               print word
               reset word
               state = 0

            eof:
               check word
               print word
               state = 2

         end case

   end case
end while
```