Estrutura de Dados

Trabalho Prático II

Prof. Josimar Viana

Maio, 2023

Processamento de texto

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.

O verificador ortográfico 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.

Exemplos de processamento

Considere a 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".
```

Exemplos de processamento

Vai gerar a 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".
```

Dicas

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.

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:

Processamento

```
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.
```

Dicas

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”).

Instruções gerais

A implementação deve atender os seguintes requisitos:

1) Funcionar corretamente com os exemplos deste documento
2) 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).
3) A localização das palavras no dicionário deve usar um algoritmo de busca binária.
4) O executável deve se chamar ortografia.
5) Processar o arquivo brascubas.txt em menos de 1 segundo (com sugestões desativadas).

Instruções gerais

4) Para ser válido, o código deve compilar e executar com sucesso; códigos que não compilarem, não executarem até o fim, travarem, encerrarem com erros de sistema (segmentation fault, bus error, no more memory, etc) ou tiverem comportamento instável terão nota zero.
5) No final da tarefa estão previstas entrevistas individuais para defesa dos projetos realizados.
6) Para medir o tempo de execução do programa, use o comando time:

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

Instruções gerais

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;

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.

Sugestão de implementação

```
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)
 escrever c em stdout
 c = ler caractere da entrada
fim enquanto
```

Sugestão de implementação

```
// 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
```