Corretor ortográfico em C++

Vinícius Crepschi, Pedro Coelho, João Victor Montefusco, Natham Corracini

10/08/2018

CIÊNCIA DA COMPUTAÇÃO

ED1 E POO

PROF. MÁRIO LIZIER E PROFA. KATTI FACETTI

Conteúdo

1. Introdução

1.1. Conceitos aplicados

O presente trabalho tem como objetivo apresentar um software de correção ortográfica, implementado em C++. Feito como integração entre as disciplinas de *Estruturas de Dados I* e *Programação Orientada a Objetos*, o trabalho combina os seguintes conceitos vistos nas duas disciplinas:

- ED1:
    1. Árvore AVL
    2. Uso de iteradores
    3. Pilha e lista da STL
    4. Manipulação de Arquivos
- POO:
    1. Estruturação em Classes
    2. Diagrama de Classes
    3. Encapsulamento
    4. Sobrecarga de Operadores

1.2. Descrição do problema

A descrição fornecida para o problema foi:

”A sua equipe foi contratada para desenvolver um corretor ortográfico. O corretor deverá permitir a verificação de erros num texto comparando as palavras no arquivo de texto com as palavras num dicionário. Palavras que não fazem parte do dicionário são potenciais erros de escrita. Assim, caso o corretor encontre uma palavra que não faça parte do dicionário, ele deverá permitir ao usuário: corrigir a palavra, ignorar o erro, selecionar uma palavra a partir de uma lista de palavras semelhantes ou adicionar a palavra no dicionário. Ao apresentar um erro para ser corrigido, o corretor deverá apresentar também ao usuário, o contexto em que o erro ocorreu (a palavra anterior e a palavra seguinte ao erro). O corretor deverá também manter uma lista dos erros encontrados no texto (corrigidos ou não). Essa lista deverá conter apenas uma entrada para cada palavra errada e deverá manter o número de vezes que o mesmo erro ocorreu no texto. O Corretor deverá ser uma classe que contenha um dicionário, um texto e a lista das palavras erradas.

A classe Texto deverá conter uma lista de palavras, o nome do arquivo original do texto e permitir carregar um texto a partir de um arquivo, percorrer o texto, palavra por palavra, alterar uma palavra e gravar o texto em um arquivo (não necessariamente o original).

A classe Dicionário deverá armazenar as palavras obtidas do arquivo ”dict.txt”e deverá permitir consulta e inclusão de palavras e fornecer uma lista de palavras semelhantes à uma determinada palavra (considerar palavras semelhantes aquelas que começam com as mesmas 2 letras). O relacionamento entre as classes Dicionário e Palavra deverá ser implementado por meio de uma árvore balanceada. Você poderá escolher entre AVL e vermelho-preta.

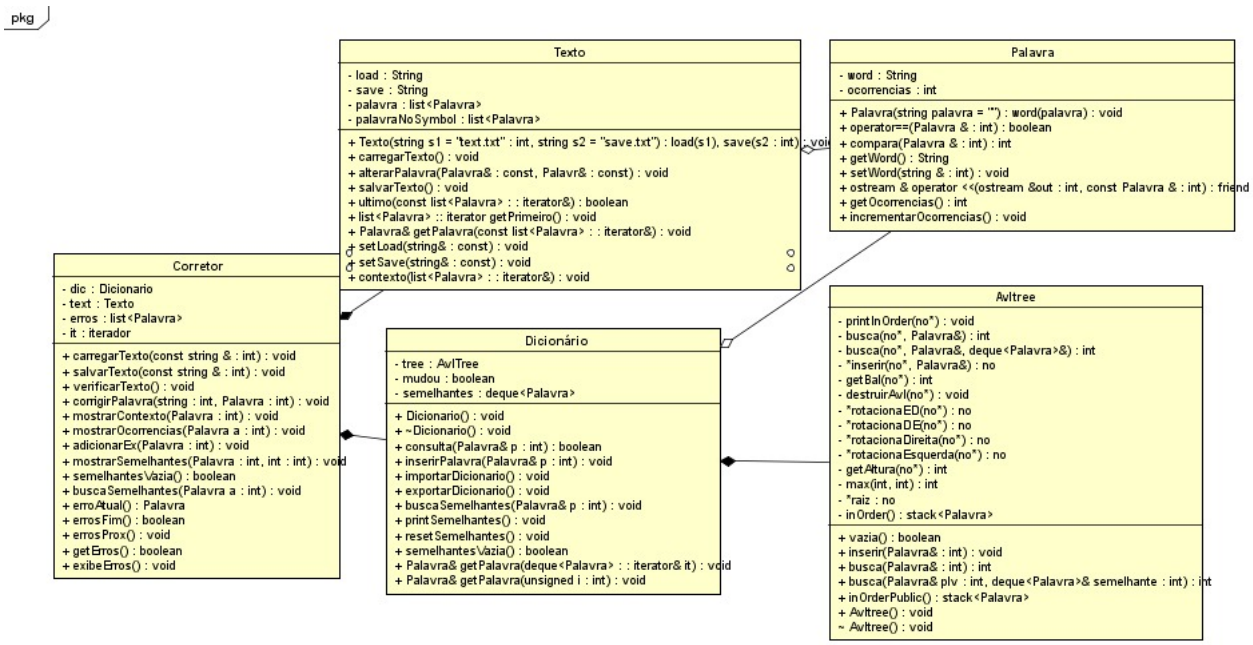
A classe Palavra deverá conter ao menos a palavra, um método que verifique se duas palavras são semelhantes e ter o operador == sobrecarregado para verificar se duas palavras são idênticas.

O sistema deverá conter pelo menos as classes Corretor, Dicionário, Texto, Palavra e Arvore. A classe corretor deverá conter métodos para a interação com o usuário e também para manipular as classes dicionário e texto.

A classe Arvore deve ter na interface apenas os métodos: vazia, insere, remove e Busca."

2. Diagrama de Classes

O diagrama de classes completo foi construído a partir do diagrama fornecido nas instruções que já explicitava o relacionamento entre as classes. Adicionamos então todos os métodos e atributos que foram necessários:



3. Fluxo de Execução

O fluxo básico de execução do programa principal é o seguinte:

1. Arquivo de dicionário é lido
2. Usuário insere nome do arquivo texto
3. Arquivo de texto é lido
4. Usuário insere sim ou não para iniciar a correção ou sair
5. Programa varre o texto em busca de erros, comparando cada palavra com as do dicionário
6. Para cada palavra errada o usuário decide dentre quatro opções
    - Adicionar palavra como exceção
    - Lista sugestões do dicionário, seguido da escolha de uma palavra para substituir
    - Corrigir manualmente a palavra inserindo uma nova
    - Ignorar o erro e prosseguir para o próximo
7. ao final da correção, usuário insere o nome do arquivo corrigido a ser exportado

4. Classes

4.1. Corretor

A classe corretor é responsável por integrar as classes Texto e Dicionário, fornecendo ao usuário uma API completa para que se possa construir no programa principal o fluxo descrito acima.

Atributos privados :

Dicionario dic Objeto da classe Dicionário.

Texto text Objeto da classe Texto.

list<Palavra> erros Lista de objetos do tipo Palavra para armazenar palavras erradas.

list<Palavra> iterator :: it Iterador para a lista de erros.

Métodos públicos :

carregarTexto Recebe string da main e chama a função do Texto para leitura do arquivo com o nome especificado.

salvarTexto Recebe string da main e chama a função do Texto para salvar o texto corrigido no arquivo de nome especificado.

verificarTexto Percorre a lista de palavras do Texto e compara cada uma com as palavras do Dicionário, inserindo em uma lista as palavras consideradas erradas.

corrigirPalavra Recebe uma string da main e uma Palavra da lista de erros .

mostrarContexto Acessa o texto e imprime as palavras anterior e posterior ao erro.

adicionarExcecao Acessa o dicionário e inclui o erro atual como exceção no arquivo do dicionário.

mostrarSemelhantes Imprime as palavras da lista de semelhantes contida no dicionário e em seguida, reseta a lista.

corrigirSemelhante Corrige a palavra no texto utilizando a palavra escolhida da lista de semelhantes.

semelhantesVazia Retorna true caso não existam palavras semelhantes no dicionário.

buscaSemelhantes Percorre o dicionário em busca de palavras que possuem as mesmas duas primeiras letras da palavra errada e as insere num deque próprio do dicionário.

erroAtual Retorna a palavra correspondente ao iterador da lista de palavras erradas do corretor.

errosFim Retorna true caso o iterador da lista de erros se encontre no fim.

errosProx Avança o iterador da lista de erros.

getErros Retorna verdade caso a lista de erros não esteja vazia.

exibeErros Imprime na tela a lista completa de palavras erradas no texto.

4.2. Árvore

A classe Árvore é uma implementação de uma árvore AVL, ou seja, possui balanceamento automatico. A interface da classe Árvore possui, conforme o requisito do trabalho, uma interface com apenas funções de inserção, busca e vazia.

Atributos e métodos Privados :

busca Procura na árvore pela Palavra inserida
busca (da sobrecarga) Particiona a palavra, pegando apenas as primeiras duas letras e procura na árvore por palavras com mesmo começo, colocando no deque de semelhantes do Dicionário.
inserir Insere recursivamente o item na árvore.
getBal Calcula o balanceamento da subarvore. Usado no inserir.
destruirAvl Desaloca memória utilizada pela árvore. Usado no destrutor da árvore
rotacionaEsquerda Realiza uma rotação para a esquerda
rotacionaDireita Realiza uma rotação para a direita
rotacionaED Rotaciona a sub-árvore A-¿esquerda para a esquerda e depois rotaciona A para a direita.
rotacionaDE Rotaciona a sub-árvore A-¿direita para a direita e depois rotaciona A para a esquerda.
getAltura Recupera o dado altura da struct nó.
max Retorna o maior dentre dois inteiros.
inOrder Exporta a árvore em ordem.
no* raiz Atributo privado que representa a struct nó, que possui nós esquerda e direita, uma Palavra e o atributo altura.

Métodos Públicos :

vazia Retorna true se a arvore se encontra vazia.
inserir Recebe uma Palavra e chama o inserir privado para realizar a inserção na árvore.
busca Recebe uma Palavra e chama a busca privada para verificar a existência do item na árvore.
busca (sobrecarregada) Recebe uma Palavra e um deque do Dicionário e busca as palavras semelhantes para inserir no deque do Dicionário.
inOrderPublic Função para exportar o dicionário percorrendo a árvore em ordem.

4.3. Palavra

A classe Palavra é responsável pelo manuseio das palavras através de uma string e o número de ocorrencias para contagem de erros, seja na comparação ou sobrecarga de operadores.

Atributos Privados :

word String para receber a palavra.
ocorrencias Int que armazena o número de vezes que um mesmo erro ocorreu.

Métodos Públicos :

Palavra Construtor que incializa ocorrencias como 0.
operator== Sobrecarga do operador == para comparar duas Palavras.
compara Baseado no std::compare para strings, usado para percorrer a árvore.
getWord Acessa a string do objeto Palavra.
setWord Acessa e modifica a string do objeto Palavra.
ostream & operator << Sobrecarga do operador de stream ¡¡ para impressão de Palavra no cout.
getOcorrencias Acessa o campo ocorrencias.

4.4. Dicionário

A classe dicionário tem como objetivo armazenar um dicionário, inserido através de um arquivo de texto, para ser usado como uma ferramenta de consulta e correções a textos inseridos pelo usuário.

Atributos Privados :

tree Objeto da classe árvore AVL onde o dicionário é armazenado e as buscas efetuadas.
mudou Variável do tipo bool que detecta se houveram inserção de novas palavras no dicionário.
semelhantes Deque do tipo Palavra para armazenar palavras semelhantes a uma determinada palavra.

Métodos públicos :

Dicionário Inicializa o atributo mudou como false e chama os construtores padrões de seus demais atributos.
consulta Efetua uma busca pela palavra passada como parâmetro no dicionário, chamando a função de busca da Árvore. Retorna true caso a palavra seja encontrada e do contrário false.
inserirPalavra Insere palavra passada como parâmetro para a função, no dicionário, chamando a função de inserir da Árvore. Além disso a função muda o valor do atributo "mudou" da classe Dicionário para True.
importarDicionario Carrega o dicionário a ser utilizado como base através de um arquivo de texto com o nome "dic.txt" presente no mesmo diretório, inserindo cada palavra encontrada na Árvore, chamando a função inserir da mesma e passando cada palavra lida do arquivo como parâmetro.
exportarDicionario Salva o dicionário em tempo de execução (armazenado dentro da Árvore) em um arquivo de texto de nome "dic.txt", desde que o dicionário tenha sido alterado desde o momento em que foi importado (mudou = True).
buscaSemelhantes Insere palavras semelhantes aquela passada como parâmetro para a função, no deque (atributo privado de Dicionário), chamando a função de busca sobrecarregada da Árvore.
printSemelhantes Imprime cada uma das palavras dentro do deque de palavras semelhantes (atributo privado de Dicionário), imprimindo apenas as 6 palavras mais semelhantes.

resetSemelhantes Apaga todas as palavras antes inseridas no deque, para que novas palavras possam ser inseridas.

semelhantesVazia Checa se o deque para armazenar palavras semelhantes está vazio, chamando a função empty da biblioteca padrão, caso o deque esteja vazio retorna True e do contrário retorna False.

getPalavra Através de um iterador do tipo deque Palavra, passado como parâmetro, retorna a Palavra correspondente aquela a qual o iterador aponta.

getPalavra (sobrecarregada) Através de um número sem sinal, passado como parâmetro, retorna a Palavra presente na posição representada por este índice no deque.

5. Texto

A classe texto tem o propósito de ler o arquivo texto e armazenar o texto em uma lista de palavras, tratando pontuação devidamente e depois inserindo de volta o texto corrigindo em outro arquivo texto.

Atributos privados:

save Armazena a string referente ao nome do arquivo a ser salvo após a correção do texto.

load Armazena a string referente ao nome do arquivo a ser carregado.

palavra Armazena palavras lidas do arquivo de texto em forma de lista, sem tratamento algum.

palavraNoSymbol Armazena palavras lidas do arquivo de texto em forma de lista, após tratamentos, por exemplo remoção de pontuação.

Métodos públicos :

construtor Inicia as strings load e save, atributos privados da classe com valores padrões: “text.txt” e “save.txt”.

carregarTexto Carrega o texto a ser corrigido através de um arquivo do tipo texto, o qual o nome é inserido pelo usuário na main, e armazena o mesmo no formato de lista (atributos private da classe).

alterarPalavra Recebe duas palavras como parâmetros, a palavra correta (advinda da seleção do usuário através de sugestões de palavras semelhantes no Dicionário) e a palavra errada (advinda da correção do texto), e troca a palavra errada no texto, pela correta.

salvarTexto Salva o texto corrigido em um arquivo de texto, com o nome inserido pelo usuário.

ultimo Recebe como parâmetro um iterador do tipo lista de Palavra e checa se a palavra a qual o iterador aponta, é a última palavra da lista, caso seja, retorna True e do contrário False.

getPrimeiro Retorna um iterador do tipo lista de Palavra que aponta para o primeiro elemento da lista.

getPalavra Recebe como parâmetro um iterador do tipo lista de Palavra e retorna a Palavra para o qual aponta.

setLoad Recebe como parâmetro uma string e faz com que o atributo load a receba (utilizado para armazenar o nome do arquivo a ser carregado, passado pelo usuário na main).

setSave Recebe como parâmetro uma string e faz com que o atributo save a receba (utilizado para armazenar o nome do arquivo a ser salvo, passado pelo usuário na main).

contexto Recebe como parâmetro um iterador do tipo lista de Palavra e imprime na tela as Palavras presentes no contexto daquela que o iterador aponta. Mais precisamente, imprime a palavra anterior e a posterior aquelas apontada pelo iterador.

6. Tratamento de exceções

O tratamento de exceções é feito pelo mecanismo try-catch e throw do próprio C++. Dentre as exceções tratadas, temos as seguintes:

1. Nome do arquivo texto inválido.
2. Dicionário não presente na pasta.
3. Dicionário aberto porém vazio.
4. Falha em salvar o arquivo.
5. Falha em salvar o dicionário.

7. Conclusão

O foco de desenvolvimento do trabalho, além de trabalhar em equipe pela primeira vez em um projeto de programação, foi o uso do C++ que mostrou-se um desafio a mais para o grupo, o que nos levou a maior pesquisa de materiais, além do fornecido em aula.

Através deste trabalho foi possível aprender de maneira mais profunda o uso do paradigma de orientação a objetos, de maneira prática, no desenvolviento de uma aplicação realista.

Outra ferramenta utilizada que exigiu certo estudo foi o Git para controle de versões, bem como o GitHub para armazenar o repositório. Aprendemos a lidar com branches, commits e pushes, de forma a manter os integrantes sempre atualizados a cada modificação no código-fonte.

Do ponto de vista didático, portanto, conclui-se que o trabalho foi de grande valia para o conhecimento do paradigma, do C++ em si e das estruturas de dados que, até então, foram estudadas de um ponto de vista teórico, focado na implementação