UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO

SCC0223 Estruturas de Dados – 2º Semestre 2020
Profa. Dra. Elaine Parros M. de Sousa
Estagiária PAE: Eliane Gniech Karasawa

Projeto II: Verificador de palavras

Data de Entrega: **11/12/2020**

Este projeto deverá ser feito em **duplas**. A entrega deverá ser realizada via *run.codes* por apenas **um integrante** do grupo, fazendo *upload* dos arquivos em "**PROJETO II 11/12"**.

Descrição do Problema

A busca por conteúdo em texto pode se basear, de maneira simplificada, na alta presença de palavras relevantes. Essa busca necessitaria de um dicionário com as palavras mais pertinentes ao domínio do tema objetivo (assunto) e o texto a ser avaliado. Existem abordagens mais aprimoradas que tentam avaliar a semântica dos trechos, por exemplo o uso de sentido figurado, mas esse não será o nosso foco.

A finalidade desse trabalho será avaliar trechos utilizando dicionários fornecidos, retornando as palavras do trecho que não estão contidas no dicionário e as palavras mais frequentes do trecho que são palavras-chave.

Objetivo do Trabalho

O trabalho tem como principal objetivo verificar trechos de texto a partir de um dicionário e retornar as **palavras não encontradas** e as **n palavras do dicionário mais frequentes** no trecho, com suas respectivas **frequências**.

O desenvolvimento do trabalho envolve as seguintes atividades principais:

1. Modelar a solução para o problema dado utilizando Estruturas de Dados não-lineares vistas em aula (de AB em diante).
2. Desenvolver uma solução para o problema dado.
3. Implementar a solução em Linguagem C, usando (obrigatoriamente) o conceito de TAD, modularização de código e Makefile para compilação e geração da aplicação executável.

Essas atividades compõem as 2 partes do trabalho, descritas a seguir.

Parte I – Modelagem da Solução (40% da nota final)

Serão avaliadas a coerência e relevância da solução e respectiva justificativa para os seguintes aspectos:

- Quais Estruturas de Dados e respectivas abordagens de implementação foram escolhidas para modelar o problema? Justifique baseado na solução implementada.
- Quais vantagens e limitações as estruturas/implementações escolhidas apresentam?
- Explique **claramente e detalhadamente** a lógica utilizada para resolver o problema. Como as estruturas (e respectivas operações básicas) são utilizadas para representar (e resolver) o problema?

Parte II – Implementação (60% da nota final)

Esta parte avalia a estratégia de implementação utilizada. Por isso, serão considerados aspectos como organização, corretude, eficiência do código e documentação interna.

A manipulação das Estruturas de Dados deve ser baseada em **TAD** e o código deve ser escrito em Linguagem C (C99, flag -Wall), compilado com **gcc**. Além disso, sua implementação deve atender os requisitos descritos a seguir.

Entrada do Programa: Serão disponibilizados 3 arquivos de teste que mostram como deverá ser a entrada do programa. Todas as palavras serão em minúsculo e não haverá caracteres especiais nem sinais de pontuação no meio do trecho. O programa deverá ser capaz de executar 5 operações:

1) **Criar dicionário:**
   - Será indicada a operação de criação na primeira linha, seguida das palavras a serem inicialmente adicionadas no dicionário, com uma palavra por linha (nenhuma ordem pode ser assumida).
   - Poderão existir até 3 dicionários ao mesmo tempo.
   - O ID do dicionário criado deverá ser sempre o primeiro disponível, dentre (1, 2 e 3). Se existe apenas o dicionário de ID 2, por exemplo, ao criar um novo, esse possuirá ID 1.
   - O fim da entrada será indicado por "#".
   - O retorno da função deverá ser "DICIONARIO <<ID do dicionário>> CRIADO".

   Tratamento de exceção:
   - Tentativa de criação de um 4º dicionário: "IMPOSSIVEL CRIAR";
   - Palavra repetida dentro da lista de palavras deve ser ignorada.

2) **Atualizar dicionário:**
   - Na primeira linha serão indicados a operação e o ID do dicionário. Em seguida, serão informadas as palavras a serem atualizadas, uma palavra por linha. Antes da palavra, um inteiro indicará inserção (1) ou remoção (0).
   - O fim da entrada será indicado por "#".
   - Retorno da inserção: "<<PALAVRA>> INSERIDA EM <<ID do dicionário>>".
   - Retorno da remoção: "<<PALAVRA>> EXCLUIDA DE <<ID do dicionário>>".

   Tratamento de exceção:
   - Caso o dicionário não exista, deverá imprimir "DICIONARIO <<ID do dicionário>> INEXISTENTE";
   - Caso a palavra a ser adicionada já exista no dicionário, apenas imprimir na saída "<<PALAVRA>> JA EXISTE EM <<ID do dicionário>>";
   - Caso a palavra a ser removida não exista no dicionário, apenas imprimir na saída "<<PALAVRA>> INEXISTENTE EM <<ID do dicionário>>".

3) **Apagar dicionário:**
   - Será indicada a operação e o ID do dicionário a ser excluído.
   - O retorno da função deverá ser "DICIONARIO <<ID do dicionário>> APAGADO".

   Tratamento de exceção:
   - Tentativa de excluir um dicionário com ID inexistente: "DICIONARIO <<ID do dicionário>> INEXISTENTE".

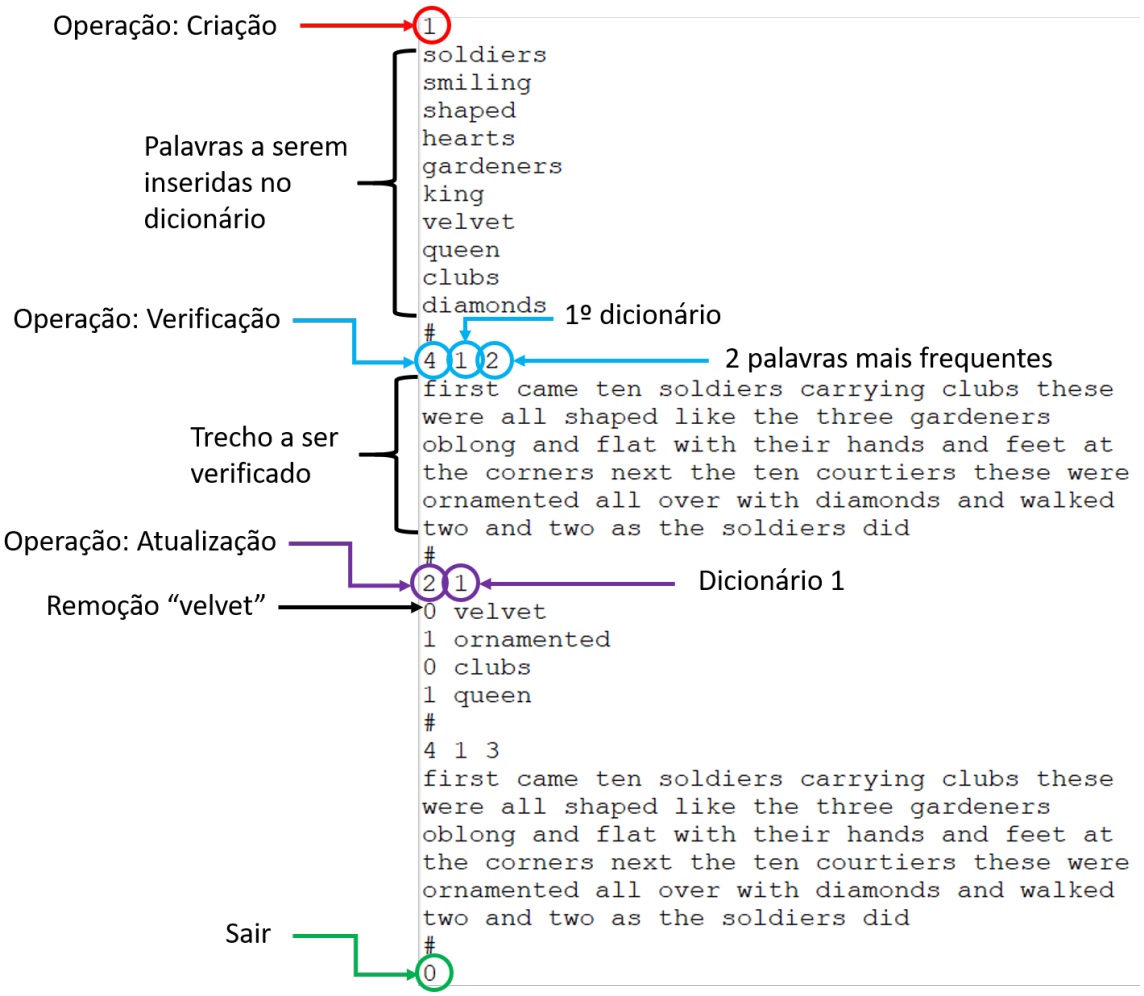
4) **Verificar texto:**
   - Na primeira linha será indicada a operação de verificação, o ID do dicionário a ser utilizado e o número **n** inteiro de palavras mais frequentes do dicionário no trecho que devem ser retornadas. O trecho a ser analisado estará **apenas** na linha seguinte.
   - O fim da entrada será indicado por "#".
   - O retorno da função deverá ser as **palavras não contidas** no dicionário, em ordem alfabética, uma por linha, e as **n palavras mais frequentes**, em ordem de frequência e alfabética, e suas respectivas **frequências**.

   Tratamento de exceção:
   - Caso o dicionário não exista, deverá imprimir "DICIONÁRIO INEXISTENTE";
   - Se o **n** for maior que o número de palavras distintas no trecho contidas no dicionário, imprimir as palavras distintas ordenadas por frequência e ordem alfabética;
   - Se **n** menor ou igual a 0, imprima "IMPOSSIVEL INFORMAR <<n>> PALAVRAS MAIS FREQUENTES";

- Caso as palavras mais frequentes possuam a mesma frequência, ordene por ordem alfabética e retorne as **n** primeiras.

0) Finalizar o caso teste.

Exemplo da Entrada



Nessa imagem, o trecho a ser avaliado abrange várias linhas para facilitar a visualização do caso teste. Nos casos teste fornecidos, o trecho estará contido em apenas uma linha.

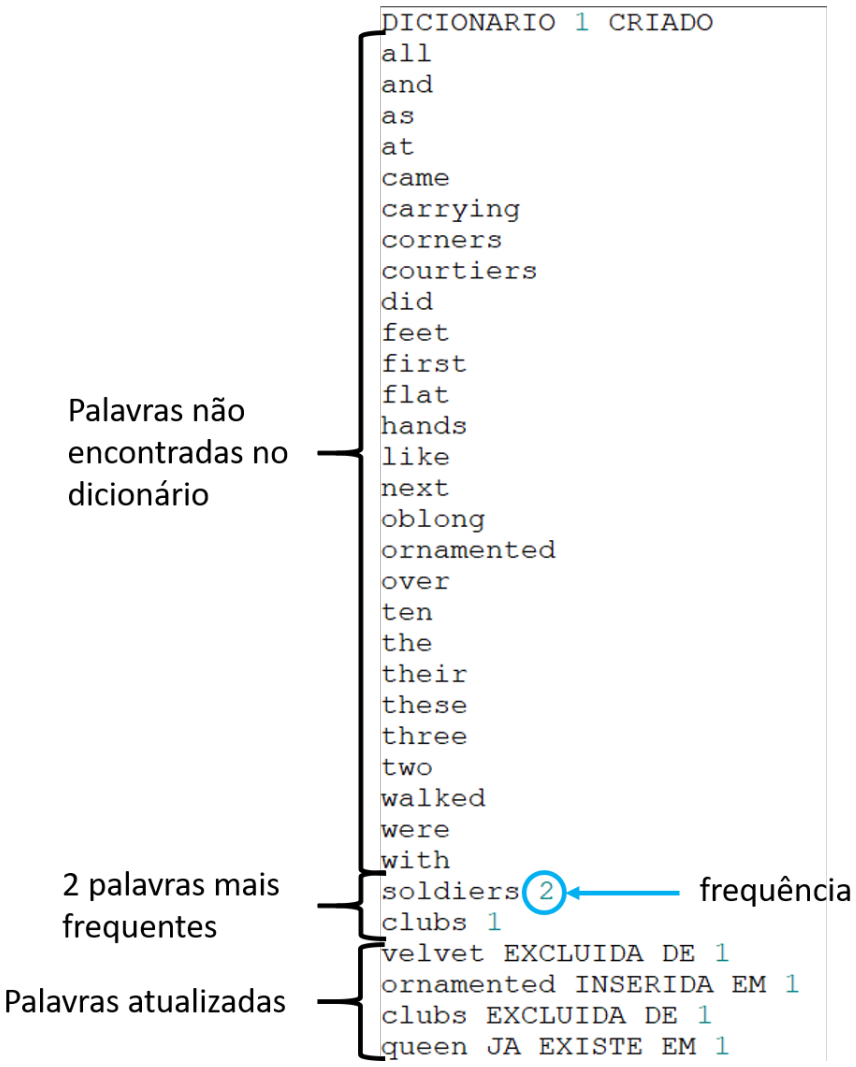
O programa deve ler um conjunto de entrada, realizar as operações requisitas e imprimir as mensagens de saída/erro ou as **palavras não encontradas** e as **n palavras mais frequentes** com suas respectivas frequências no caso da operação (4).

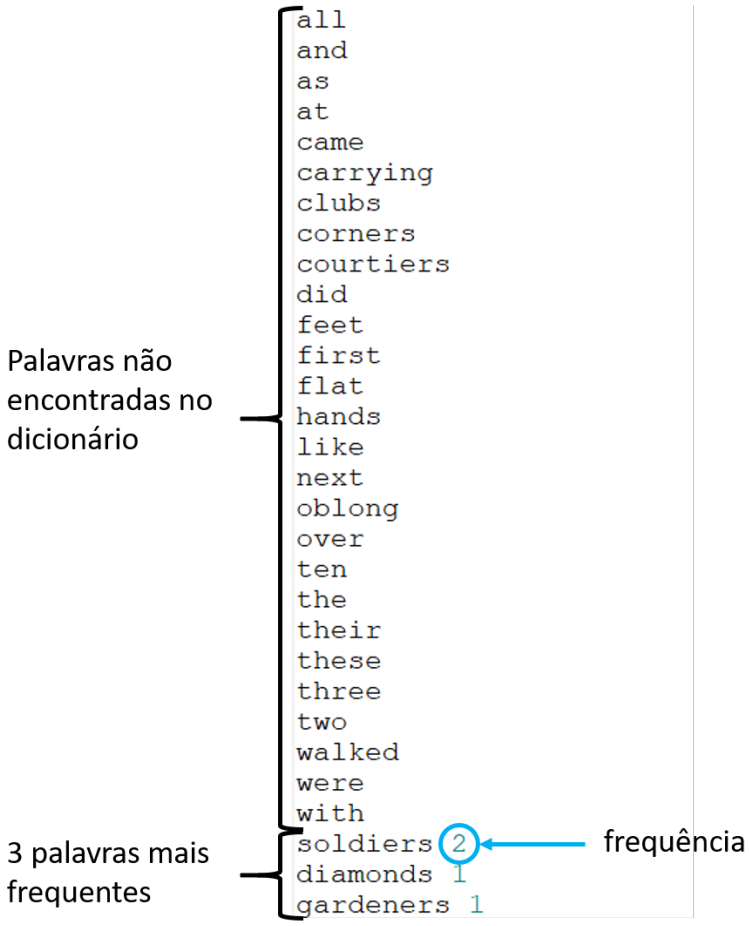
Saída do Programa: O programa deve apresentar como saída a cada vez que a verificação de texto for requisitada:

1. as **palavras não contidas** no dicionário em ordem alfabética;

2. as **n palavras mais frequentes** do trecho, com suas respectivas frequências, também em ordem alfabética.

Exemplo de Saída





Vale ressaltar que os arquivos de teste fornecidos têm como objetivo apenas apresentar o formato OBRIGATÓRIO para a entrada de dados e direcionar os testes iniciais. Na correção, seu programa será testado nos conjuntos disponibilizados e em conjuntos diferentes. Portanto, teste seu programa em conjuntos elaborados pelo próprio grupo.

Entrega do Trabalho

Faça um **relatório** de no máximo **5 páginas** com todas as informações e requisitos apresentados nas Partes I e II. Sobre a Parte II, inclua no relatório apenas a informação necessária para compilação e execução do seu programa, indicando quais são os arquivos dos TADs, do programa principal, e demais arquivos criados. Não inclua código no relatório.

O arquivo ***.zip*** contendo o **relatório** (em **pdf**), o **Makefile** e **todo código fonte** necessários para compilar e executar o programa, incluindo arquivos.c (*main* e outros) e arquivos.h deve ser enviado via *run.codes* em **Projeto II – 11/12**. Não serão aceitos/considerados arquivos executáveis.

O trabalho poderá ser entregue até **11/12/2020**, 23:59h. Trabalhos submetidos em datas posteriores terão 1,0 ponto de desconto por dia de atraso. **Apenas um integrante da dupla deve submeter o trabalho no *run.codes*.**

Apêndice Leitura de Arquivo de Entrada

Para executar seu código em casos teste muito grandes, pode ser necessário passar diretamente o arquivo de entrada. Para fazer isso, altere a linha **run** do seu Makefile para

```
./executavel <arquivo_de_entrada
```