UFV

SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Trabalho 1 - AEDS 1
Dicionário utilizando listas encadeadas

Alice Ladeira Gonçalves [5065]
Breno Júnio de Oliveira Gomes [5085]
Jeiverson Christian Ventura Miranda dos Santos [5089]

Florestal - MG
2022

SUMÁRIO

1. Introdução ……3
2. Organização ……3
3. Desenvolvimento ……..4
3.1 TAD's: Item, Lista de Linhas, Palavra ……4
3.2 TAD Lista de Palavras ……4
3.3 TAD Dicionário……..4
3.4 Main, makefile e texto.txt ……..5

4. Resultados ……..6
5. Conclusão ……6
6. Referências ……6

1. Introdução

O objetivo desse trabalho foi construir um dicionário a partir de um arquivo de texto como entrada e apresentar como saída uma lista, ordenada alfabeticamente, contendo as palavras que apareceram no texto. Sendo assim, a saída contém apenas as listas necessárias, caso não apareça nenhuma palavra com determinada inicial, a lista dessa inicial não será criada e, consequentemente, não aparecerá na saída.

A fim de atender as instruções pedidas, a abordagem utilizada para a construção do dicionario foi a utilização de listas encadeadas, as quais permitem a utilização de memória de forma dinâmica, ou seja, conforme surge a necessidade. Tal estrutura de dados é a que proporciona melhor eficiência, visto que as informações acerca de quais listas serão criadas, bem como seu respectivo tamanho são mutáveis de acordo com o arquivo a partir do qual o programa vai operar.

2. Organização

A Figura 1 apresenta a organização do projeto, nela é possível visualizar todos os arquivos que o compõe. Na pasta “TP1-Dicionario”, está a implementação do projeto separada na pasta arquivos .

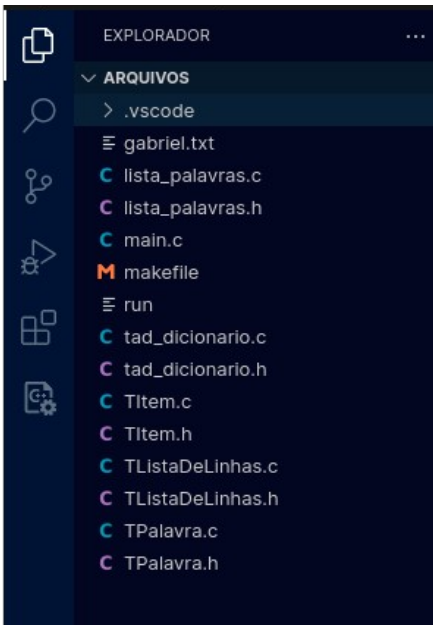

Figura 1 – Organização do projeto

3. Desenvolvimento

De acordo com a interpretação do grupo acerca das especificações do trabalho prático, os componentes do projeto devem ser encapsulados, uma parte depende da outra para ser realizada.

Isso se explica da seguinte forma: o dicionário é a estrutura mais externa, ela depende diretamente da lista de palavras para a execução de suas operações, a qual depende das palavras e número de linhas. Segue a explicação mais detalhada de cada módulo do projeto.

3.1 TAD's: Item, Lista de Linhas, Palavra

O primeiro TAD a ser criado foi o do Item, tendo dois arquivos ("TItem.h" e "Titem.c). Esse tipo é o responsável por fazer a operação de guardar um número inteiro, que é determinada linha do arquivo de texto. Sendo assim, cada número de linha é um item na lista de linhas.

Em seguida, temos o TAD Lista de Linhas, o qual recebe o arquivo "TItem.h" em seu arquivo TListaDeLinhas.h. As operações atribuídas à esse tipo são de: fazer lista de linhas vazia, inserir linha na lista e imprimir lista de linhas.

Por fim, esse módulo mais interno do projeto contém o TAD "TPalavra", que inclui o "TListadeLinhas.h" em "TPalavra.h", uma vez que cada palavra possui sua própria lista de linhas. Isso significa que, se uma palavra aparece em mais de uma linha, o TAD TListaDeLinhas é responsável por guardar todas as ocorrências dela. As operações pelas quais o TPalavra se encarrega são: criar palavra, nomear palavra, retornar palavra nomeada e inserir palavra. Em conjunto elas realizam a função de criar uma palavra vazia, com os campos a serem preenchidos, receber uma palavra e a linha, e a função de retornar a palavra é usada para imprimir essa palavra.

3.2 TAD Lista de Palavras

Lista de palavras é o TAD o qual recebe o TAD TPalavra para formar listas, agrupando aquelas com inicial em comum. As operações desse tipo são: criar lista de palavras, verificar se a lista é vazia, inserir palavra, verificar palavra, remover uma palavra dada, remover palavra do final, contar número de palavras de uma lista e imprimir lista de palavras.

3.3 TAD Dicionário

O tipo dicionário é o mais externo e precisa dos tipos supracitados para realizar suas operações. Pode resumir o Dicionário como sendo uma lista encadeada de listas de

palavras. Cada célula dessa lista possui uma lista de palavras e o ponteiro para a próxima lista. As operações desse TAD consistem em: inicializar dicionário, verificar se ele é vazio, inserir uma lista, construir o dicionário e por fim, imprimí-lo.

A operação mais importante desse tipo abstrato são as de inserção e de construção do dicionário. Notamos que, apesar de as palavras dentro de cada lista de palavras não precisarem estar alfabeticamente ordenadas, as listas em si precisavam, ou seja, caso existam as listas: “Lista de J”, “Lista de A” e “Lista de B”, elas necessariamente precisam se encadear na seguinte ordem:

|Lista de A| ptr_proximo |→ |Lista de B| ptr_proximo |→ |Lista de J| ptr_proximo |→ NULL

Essa ordenação é feita na operação “insere_lista”, a qual percorre o dicionário fazendo verificações. Se ao percorrer não encontrar nenhuma lista com letra inicial maior, ela será inserida no final, e caso encontre uma maior que ela, será inserida antes da maior.

Cabe ressaltar que a ordenação pôde ser dessa forma porque, em Linguagem C, comparar dois caracteres é o equivalente a compara seus números correspondentes na Tabela ASCII e esses, por sua vez, estão ordenados.

Ademais, a operação de construção vai organizar as palavras e suas respectivas linhas em listas correspondentes a inicial da palavra, respeitando condições de já existir uma lista daquela palavra ou não, tendo assim que criar uma nova lista para a palavra nova.

3.4 Main, makefile e texto.txt

Main é a função principal do projeto, ela vai chamar as funções necessárias para realizar as operações que o usuário decidir realizar com base nas opções fornecidas pelo menu.

O arquivo texto.txt é aquele que será usado como entrada para gerar o dicionário na saída e o makefile foi feito para compilar todos os arquivos de forma mais efetiva, tendo em vista que seria demorado digitar o nome de todos os arquivos sempre que quiséssemos compilar o projeto.

4. Resultados

Os possíveis caminhos de utilização do dicionário existem com base nas escolhas feitas pelo usuário ao interagir com o menu. Abaixo veremos as operações disponíveis para o usuário:

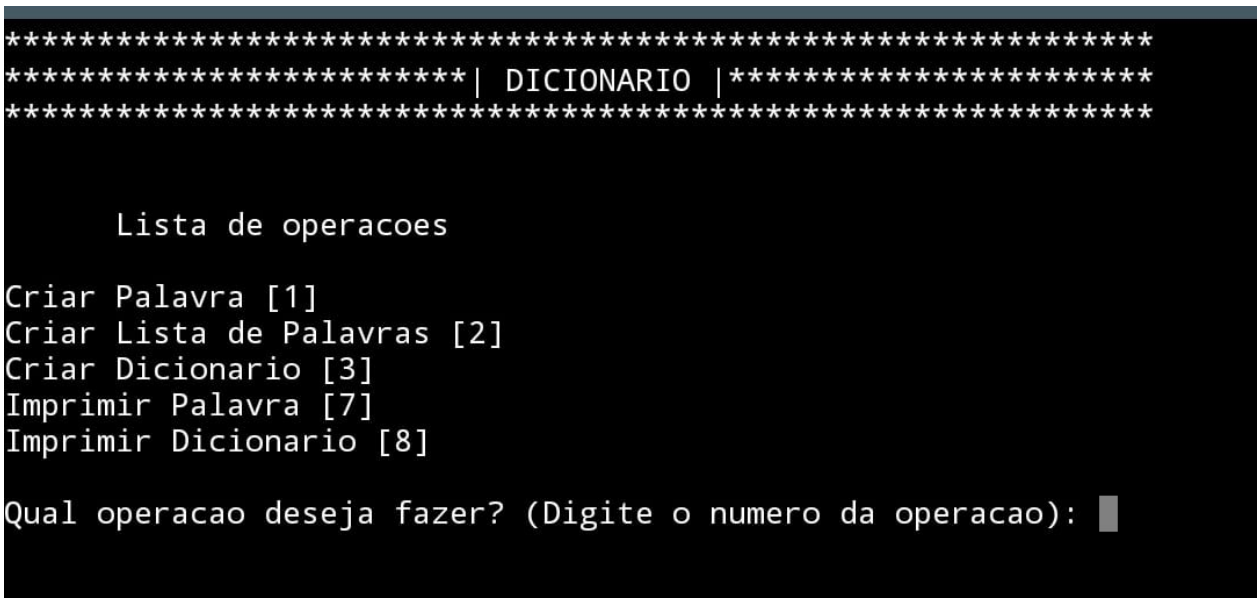

Figura 2 – Menu inicial

5. Conclusão

De forma geral, os resultados obtidos foram coerentes com a proposta recebida. O projeto precisou de manutenções até chegar na versão final desejada, o que já era esperado, tendo em vista que estamos trabalhando com um vários arquivos e uma estrutura de dados complexa. Também teria sido possível realizar esse trabalho de outras formas e até mesmo com variações dessa estrutura de dados, como o uso da lista duplamente encadeada.

No entanto talvez fosse mais difícil a construção do código pelo nosso nível de conhecimento acerca de estruturas de dados ou mesmo menos eficiente.

6. Referências

[1] ZIVIANI, Nivio. Projeto de Algoritmos com Implementações em Pascal e C. 4ª Edição. São Paulo: Pioneira, 1999.

[2] BACKES, André. Linguagem C: completa e descomplicada. Rio de Janeiro: Elsevier, 2013.

[3] SILVA, Thais Regina. Algoritmos e estrutruras de dados I (CCF211): Listas Encadeadas(Cap03 – Seção 3.1 – Ziviani). 03 out. 2022. Apresentação do Power Point. Disponível em:

https://ava.ufv.br/pluginfile.php/488027/mod_resource/content/1/Aula4_Listas2-2022.pdf . Acesso em 04 out. 2022.

[4] Visual Studio Code. Disponível em: <https://code.visualstudio.com/download>. Acesso em 15 out. 2022.

[5] Cxxdroid. Disponível em: <https://play.google.com/store/apps/details?id=ru.iiec.cxxdroid>. Último acesso em 15 out. 2022.