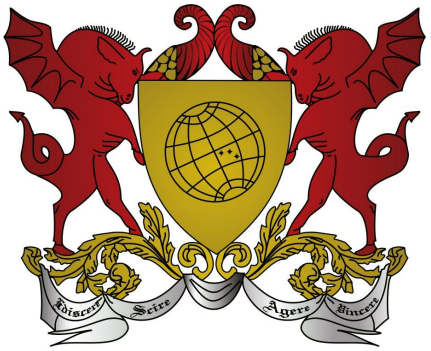


Universidade Federal de Viçosa

Trabalho Prático - Algoritmos e Estrutura de Dados: Criação de Dicionário usando Listas Encadeadas e Alocação Dinâmica

Danilo Matos de Oliveira - 5073

Guilherme Broedel Zorzal - 5064

Álvaro Gomes da Silva Neto - 5095

FLORESTAL – MINAS GERAIS

2022

1. INTRODUÇÃO

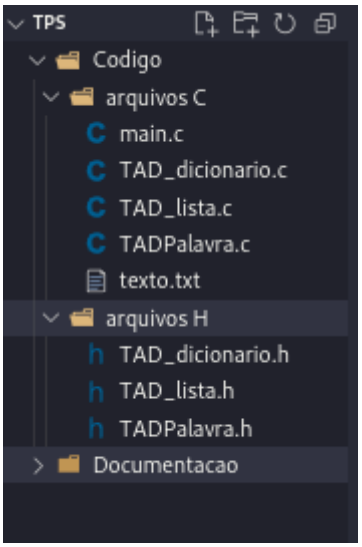
O objetivo do trabalho apresentado é a utilização e prática das estruturas de dados ensinadas em sala de aula. Para tal, foi executada a criação de um Dicionário na Linguagem C, o qual foi feito com o uso de listas encadeadas.

O Dicionário em questão irá ler de um arquivo externo com palavras e irá classificá-las de acordo com a sua primeira letra, para assim terminar a construção do dicionário. Desta forma, o documento a seguir apresenta o desenvolvimento do projeto e também as interações que o usuário poderá ter com o projeto.

2. ORGANIZAÇÃO

Na Figura 1 é possível visualizar a organização do projeto, de forma que na pasta ‘arquivos C’/ está o escopo das funções do projeto, juntamente com a implementação do arquivo main, e na pasta ‘arquivos H’/ estão presentes a implementação dos TADs solicitados para a elaboração do Trabalho prático, juntamente com a estrutura de dados, lista encadeada, solicitada.

Figura 1 - Repositório do projeto.



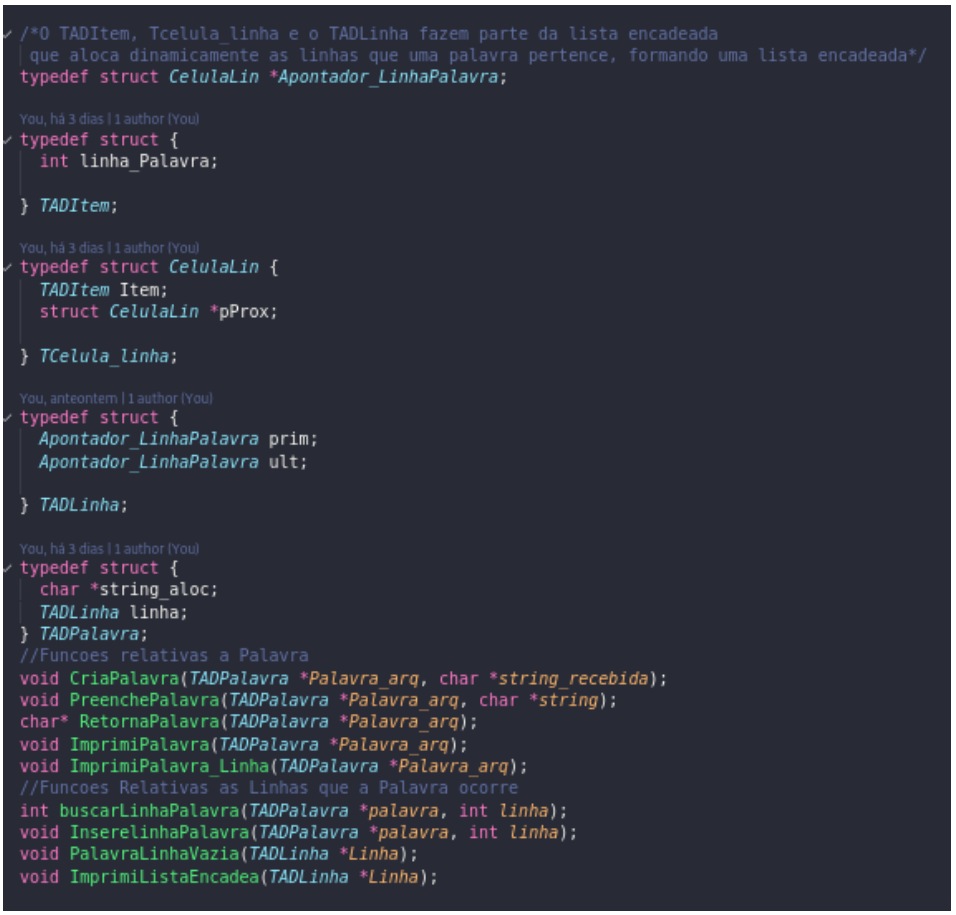
Fonte: Elaborado pelo Autor

Para executar o projeto, compilamos todos os arquivos .c para gerar um binário de execução e depois executamos esse binário.

3.DESENVOLVIMENTO

3.1. Tipo Abstrato de Dado - Palavra

Figura 2 - Implementação do TADPalavra.h



```c
/*O TADItem, Tcelula_linha e o TADLinha fazem parte da lista encadeada
 que aloca dinamicamente as linhas que uma palavra pertence, formando uma lista encadeada*/
typedef struct CelulaLin *Apontador_LinhaPalavra;

You, há 3 dias | 1 author (You)
typedef struct {
  int linha_Palavra;

} TADItem;

You, há 3 dias | 1 author (You)
typedef struct CelulaLin {
  TADItem Item;
  struct CelulaLin *pProx;

} TCelula_linha;

You, anteontem | 1 author (You)
typedef struct {
  Apontador_LinhaPalavra prim;
  Apontador_LinhaPalavra ult;

} TADLinha;

You, há 3 dias | 1 author (You)
typedef struct {
  char *string_aloc;
  TADLinha linha;
} TADPalavra;
//Funcoes relativas a Palavra
void CriaPalavra(TADPalavra *Palavra_arq, char *string_recebida);
void PreenchePalavra(TADPalavra *Palavra_arq, char *string);
char* RetornaPalavra(TADPalavra *Palavra_arq);
void ImprimiPalavra(TADPalavra *Palavra_arq);
void ImprimiPalavra_Linha(TADPalavra *Palavra_arq);
//Funcoes Relativas as Linhas que a Palavra ocorre
int buscarLinhaPalavra(TADPalavra *palavra, int linha);
void InserelinhaPalavra(TADPalavra *palavra, int linha);
void PalavraLinhaVazia(TADLinha *Linha);
void ImprimiListaEncadea(TADLinha *Linha);
```

Fonte: Elaborado pelo Autor

Na figura acima, conseguimos ver a implementação da unidade mais básica do código. Este Tipo Abstrato de Dado é o Responsável por alocar na memória todas as palavras que serão lidas do arquivo de texto, e também é responsável por alocar dinamicamente a lista de linhas nas quais a palavra aparece. As funções implementadas se dividem em dois tipos, aquelas referentes apenas ao TADPalavra e aquelas relativas ao TADLinha, que é uma lista encadeada que fica dentro do próprio TADPalavra.

3.2. Tipo Abstrato de Dado - Lista de Palavra

Figura 3 - Implementação do TAD_lista.h

```
typedef struct Celula *Apontador_celula_lista;
...
typedef struct Celula
{
    TADPalavra palavra;
    struct Celula *proximo; // Apontador_celula_lista para o prox
} Tcelula_lista;

...
typedef struct
{
    Apontador_celula_lista primeiro;
    Apontador_celula_lista ultimo;
    int tam;
} Tlista;

void cria_lista(Tlista *lista);
int leh_vazia(Tlista *lista);
void insere_palavra_linha(Tlista *lista, int linha, char *palavra);
void remove_escolhida(Tlista *lista, char *palavra);
void remove_final(Tlista *lista);
Apontador_celula_lista verifica_palavra(Tlista *lista, char *palavra);
int numero_palavra(Tlista *lista);
void imprimi_lista(Tlista *lista);
```

Fonte: Elaborado pelo Autor

Na figura acima, conseguimos ver a implementação do Tipo Abstrato de Dados responsável por encadear os elementos do TADPalavra, permitindo a criação de listas. As funções implementadas permitem a criação, manipulação e controle das listas que estão sendo trabalhadas. Este TAD também é aquele que permite a conexão entre o TADPalavra e o TAD_dicionario.

3.3. Tipo Abstrato de Dado - Dicionário

Figura 4 - Implementação do TAD_dicionario.h

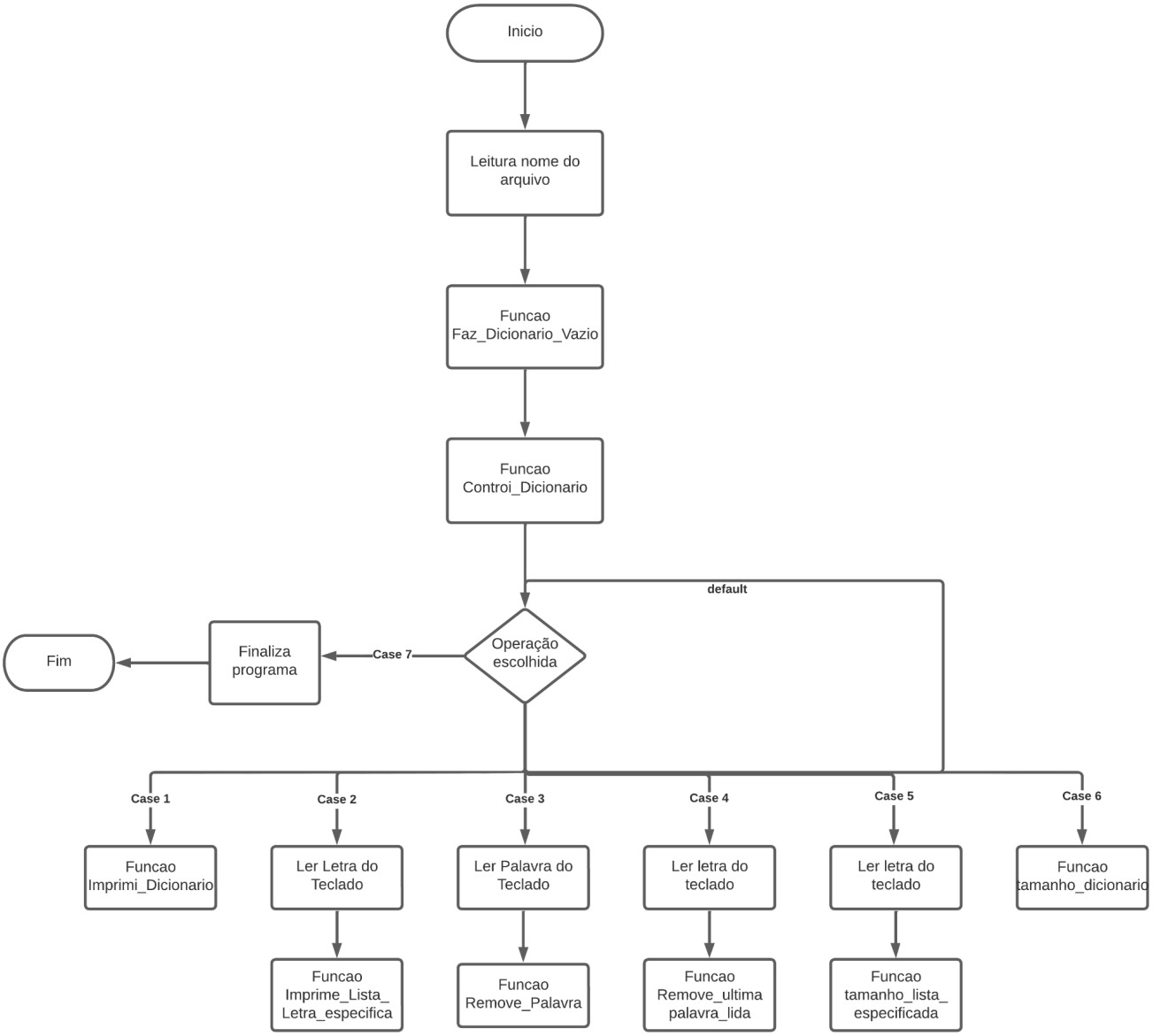
```c
typedef struct CelulaDicionario* Apontador_Celula_Dicionario; //Ponteiro pra TCelula
//Apontador_Celula_Lista

You, há 3 dias | 1 author (You)
typedef struct CelulaDicionario {
    char letra_lista; //Mostra a letra que a lista atual comporta;
    Tlista lista; //conferir *
    struct CelulaDicionario* Prox_celula_lista; // Apontador pProx; (prÂ³ximo item)
} TCelulaDicionario;

You, há 3 dias | 1 author (You)
typedef struct {
    Apontador_Celula_Dicionario pPrimeira_Lista;
    Apontador_Celula_Dicionario pUltima_Lista;
} TDicionario;

int Faz_Dicionario_Vazio(TDicionario* pDicionario);
int Imprime_Lista_Letra_Especifica(TDicionario* pDicionario, char letra);
int Imprime_Dicionario(TDicionario* pDicionario);
int Dicionario_e_vazio(TDicionario* pDicionario);
int Constroi_Dicionario (TDicionario* pDicionario, char* nome_arquivo_txt);
int Encadeia_Celula_Dicionario(TDicionario* pDicionario, char* string, char letra, int linha);
int Remove_ultima_palavra_lista(TDicionario* pDicionario, char letra_lista);
int Remove_palavra_especifica(TDicionario* pDicionario, char* string);
int Remove_celula_dicionario(TDicionario* pDicionario);
int Tamanho_lista_especifica(TDicionario* pDicionario, char letra_lista);
int Tamanho_dicionario(TDicionario* pDicionario);
```

Fonte: Elaborado pelo Autor

Na figura acima, conseguimos ver a implementação do Tipo Abstrato de Dados que faz toda a manipulação das funções anteriormente criadas tanto nos TAD's Lista e Palavra, para que o programa ocorra da forma mais eficiente possível. Além disso, as funções presentes neste TAD são aquelas que estão presentes no main do código implementado.

3.4. Execução do Main.c

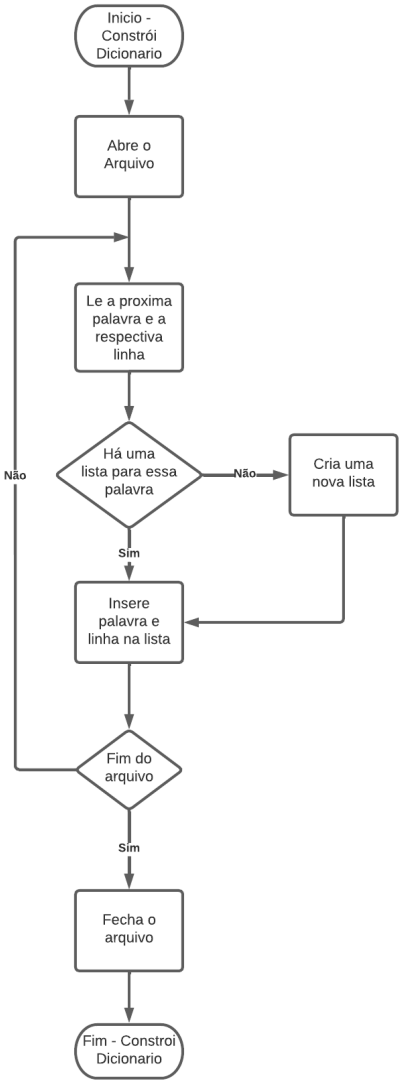
Figura 5 - Fluxograma de funcionamento do main.c



Fonte: Elaborado pelo Autor

No fluxograma acima, utilizado para abstrair as linhas de programação, percebe-se o funcionamento geral do código, e a forma como as funções são rodadas em sequência. Além disso, há também o switch case na parte inferior, que é uma das principais funções, pois permite a interação com o usuário para que ele escolha a opção desejada.

Figura 6 - Fluxograma de funcionamento da função Constroi_Dicionário

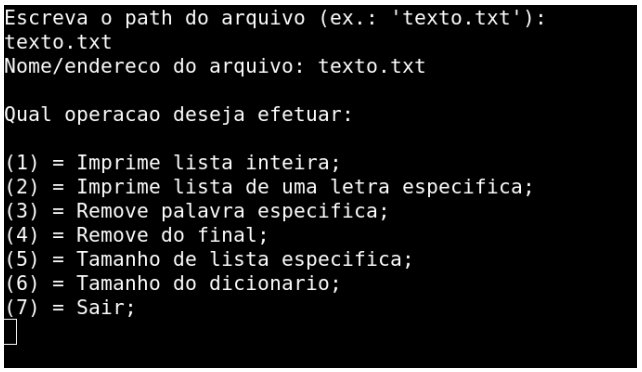


Fonte: Elaborado pelo Autor

Para explicitar melhor a função Constroi_Dicionario, que é uma das principais funções do projeto, o diagrama acima demonstra como é o seu funcionamento e também as suas principais condicionais, além de que é importante lembrar de que esta função chama várias outras funções que estão presentes dentro dos arquivos TAD_lista e TADPalavra.

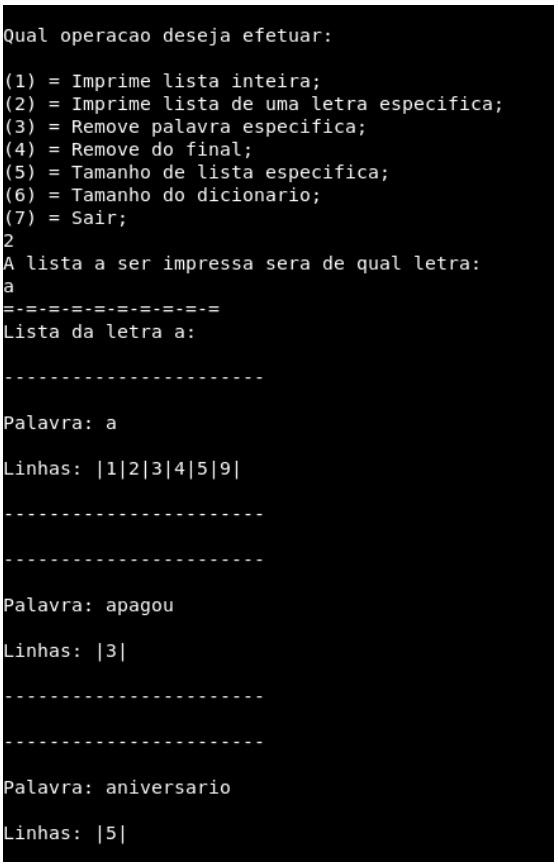
4.RESULTADOS

Figura 7 - Menu Principal do Código



Fonte: Elaborado pelo Autor

Figura 8 - Exemplo de utilização de função



Fonte: Elaborado pelo Autor

Conforme pode ser visualizado nas imagens acima, o programa executa as funções, de forma a exibir um menu de interação com o usuário. Desta forma, permite-se que o usuário interaja com o programa por meio da escolha de funções previamente definidas pelo programador no menu acima.

5.CONCLUSÃO

Sendo assim, este projeto foi realizado utilizando conceitos como Tipos Abstratos de Dados, Listas Encadeadas e Manipulação de Arquivos, nos permitindo ter uma grande capacidade prática nestes conteúdos.

Conforme a implementação do dicionário foi realizada, percebeu-se um aumento significativo em nossa capacidade de desenvolver projeto de algoritmos, pois é notável a dificuldade que um trabalho em grupo possui, já que não se trata apenas do seu próprio código, pois o essencial é a integração do seu código com a sua respectiva equipe.

6.REFERÊNCIAS

ZIVIANI, Nivio et al. **Projeto de algoritmos: com implementações em Pascal e C**. Luton: Thomson, 2004.

CORMEN, Thomas. **Desmistificando algoritmos**. Elsevier Brasil, 2017.