# Relatório Trabalho 1 de Estrutura de Dados

*Professora: Patrícia Dockhorn Costa*

*Período: 2022/1*

*EDCare: Ed Cuidando de Idosos!*

*Alunos: Marlon Moratti do Amaral e Gabriel Braga Ladislau*

# **Introdução:**

Para esse trabalho o problema é implementar um sistema que consiga analisar dados sensoriais como entrada e interpretar esses dados para gerar as saídas.

Começamos o trabalho pensando em como estruturar nosso sistema, para isso fizemos uma espécie de mapa com os respectivos *TAD’s* e suas funções, algo básico porém pratico para nos dar um caminho para seguir. Segue a foto do mapa:

Com o mapa para nos ajudar começamos a implementar nosso sistema, e usamos o *GitHub* para poder trabalhar com mais eficiência.

# **Implementação:**

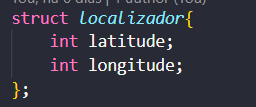
Para começar a implementação, fizemos *Bottom Up* ou seja, começamos pelo baixo nível para chegar até a *main* que no caso é o mais alto nível.

Definimos que a lista seria genérica ou seja heterogênea, ela serve tanto para cuidadores quanto para idosos.

Usamos a biblioteca *assert.h* para verificar caso os ponteiros usados nas funções do programa estão devidamente alocados e diferentes de *NULL*. Caso esteja *NULL* a função *assert()* para a execução do programa com erro (0 = erro).

***TAD Localizador:***

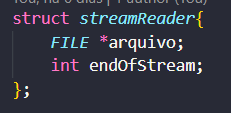
Começamos então a desenvolver o *TAD* de Localizador, que carrega os dados de latitude e longitude dos indivíduos. O *struct* implementado ficou assim:



No arquivo *localizador.h* temos as funções de criar localizador (aloca espaço), recupera e modifica latitude e longitude, deleta localizador (*free*), calcula distância entre dois localizadores e uma função de *Debug* para confirmar se o localizador recebe e guarda as informações corretamente. As funções foram implementadas no .c.

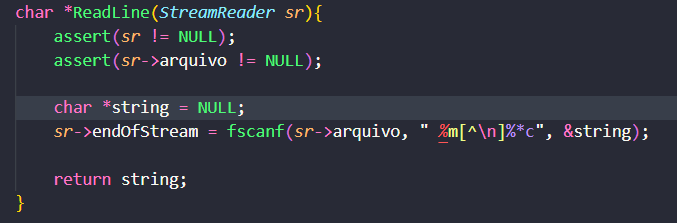
***TAD StreamReader :***

Com o Localizador pronto, começamos a fazer o *TAD* que leria arquivos, chamamos de *StreamReader.* Esse *TAD* na realidade era composto por outro chamado *Filemanager* porém para seguir o mapa de desenvolvimento suas funções foram incluídas nos *TAD’s StreamReader e StreamWriter.* Para ele não carregar simplesmente o arquivo a ser lido fizemos um struct que carrega caso o arquivo tenha chegado ao final (*endOfStream = 1*). Segue o *struct:*



Nesse TAD foram desenvolvidas funções de criar *StreamReader* recebendo o caminho do arquivo, recupera *end of stream* , deleta *StreamReader* (*free*) e *ReadLine*.

A função mais importante nesse TAD é a *ReadLine.* A função lé o arquivo alocando diretamente o tamanho da linha na *string.*



O *%m* no *scanf* serve para fazer um *malloc* direto da *string,* facilitando nosso trabalho e não precisando usar um *buffer.*

***TAD StreamWriter:***

O *TAD StreamWriter* tem uma função parecida com o *StreamReader,* porém neste caso ele carrega o arquivo que devemos escrever (as saídas do programa). Ele não tem nenhuma peculiaridade. A sua struct carrega somente o arquivo para escrever. Segue o StreamWriter.c :

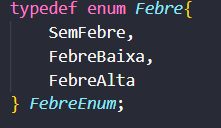


***Febre Enum***

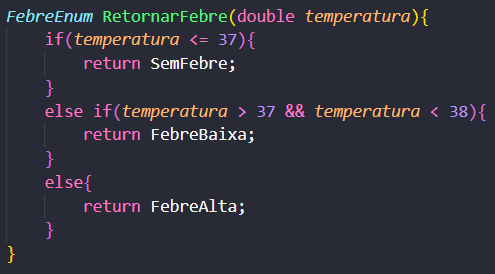
Para tratar a febre ou não do idoso, fizemos um *enum* que nesse caso facilita o trabalho de escrever qual o estado de febre do idoso.

A definição de *enum* é: “É um tipo de dado definido pelo usuário que define uma variável que vai receber apenas um conjunto restrito de valores. Na realidade, um *enum* é um conjunto de valores inteiros representados por identificadores”.

Ou seja com esses identificadores nós conseguimos colocar se o idoso está com febre baixa, alta ou sem febre sem precisar atribuir zeros e uns, utilizando algo mais alto nível que se assimila a linguagem natural. Segue o tipo definido *FebreEnum,* do arquivo *FebreEnum.h* abaixo:

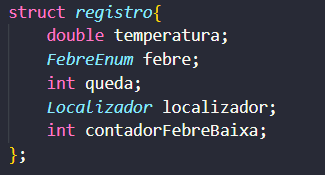


Ele têm somente uma função, para calcular a febre dependendo da temperatura e atribuir ao registro do idoso (disponível em *FebreEnum.c*):



***TAD Registro:***

Para guardar todas as variáveis necessárias dos idosos do sistema chegamos a conclusão de estabelecer um tipo abstrato de dados, que guardasse todos esses dados dentro do *TAD* de idosos. Para isso que o nosso *TAD* de registro serve. Suas funções foram feitas pensando em como iriamos interpretar os dados recebidos da entrada, de tal maneira que a cada iteração ou nova linha lida, o programa guardasse os dados daquele momento e a interpretação destes fosse feita logo após, assim a cada rodada o registro é mudado e temos diferentes dados nele. Segue o *struct (disponível em Registro.c)* :



As funções contidas no .c somente atualizam ou retornam alguma variável de registro, além da cria, deleta e uma de *Debug.*

***TAD Cuidador***