# Relatório Trabalho 1 de Estrutura de Dados

*Professora: Patrícia Dockhorn Costa*

*Período: 2022/1*

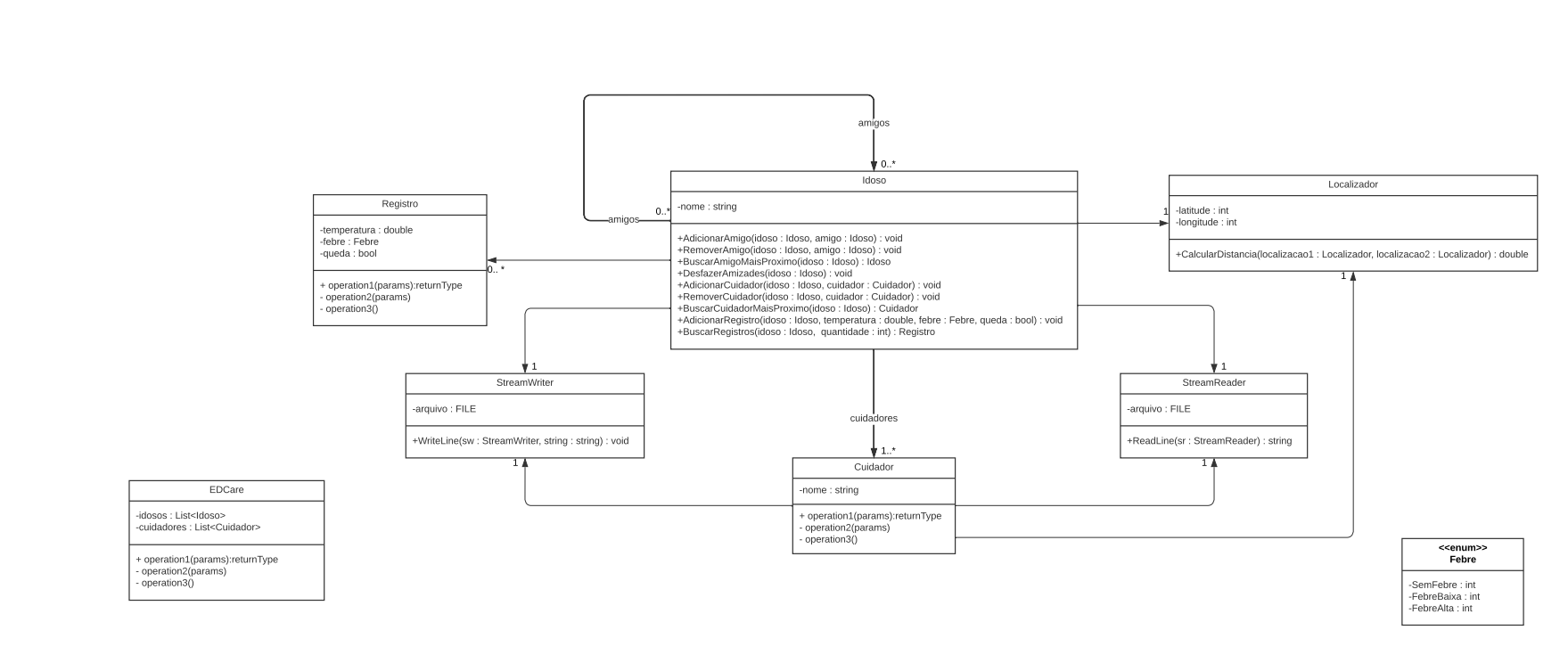
*EDCare: Ed Cuidando de Idosos!*

*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:**

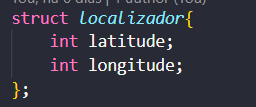
Para começar a implementação, fizemos *Bottom Up* ou seja, começamos pelo baixo nível para chegar até a *main* que no caso é o mais alto nível.

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:***

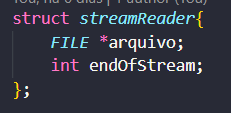
Começamos então a desenvolver o *TAD* de Localizador, que carrega os dados de latitude e longitude dos indivíduos. O *struct* implementado ficou assim:



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.

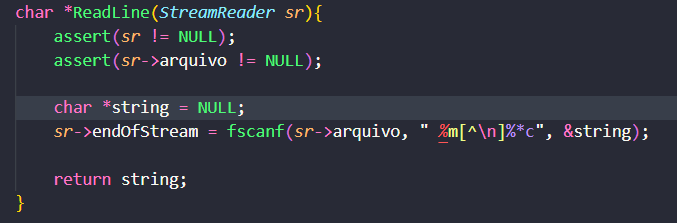
***TAD StreamReader :***

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 :

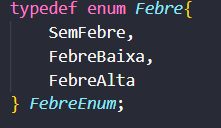


***Febre Enum***

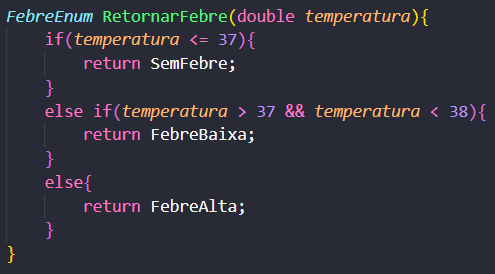
Para tratar a febre ou não do idoso, fizemos um *enum* que nesse caso facilita o trabalho de escrever qual o estado de febre do idoso.

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:

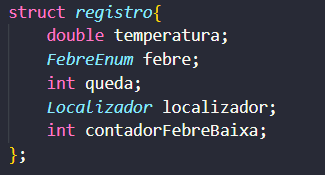


Ele têm somente uma função, para calcular a febre dependendo da temperatura e atribuir ao registro do idoso (disponível em *FebreEnum.c*):



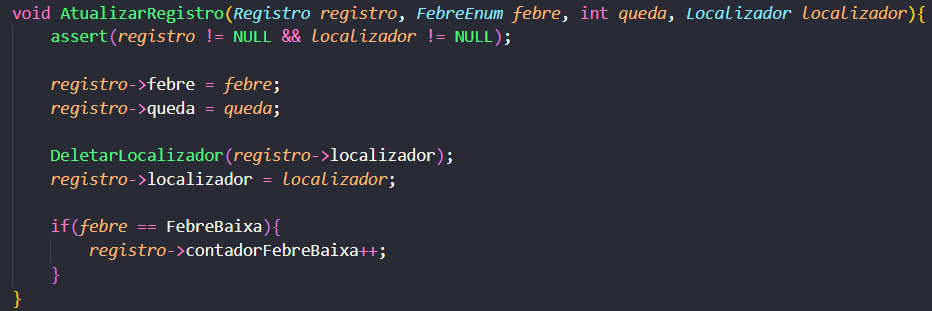
***TAD Registro:***

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.*

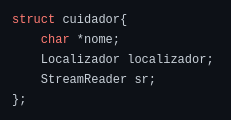
Temos também a Atualiza Registro:



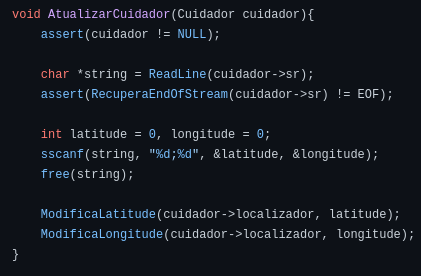
Ela pega os novos dados e mudam o registro do idoso (o *Localizador* antigo é liberado da memória e um novo é criado).

***TAD Cuidador***

O *TAD* de cuidador serve para administrar o Cuidador, peça fundamental do sistema EdCare. Como temos uma mudança de variáveis por ciclo, o s*truct cuidador* não tem tanto “mistério”, segue ele abaixo:

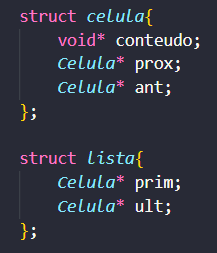


De peculiaridade a função de criar cuidador usa de funções da biblioteca *string.h* , a *strdup() e sprintf().* A primeira faz uma alocação dinâmica de um *char\** para outro, fazendo uma cópia. A segunda é usada para escrever em uma *string,* nesse caso é usada para escrever o caminho onde o arquivo de certo cuidador está localizado. Outras funções da *string.h* usadas são a *strcmp()* que compara duas *strings*, (retornando 0 se iguais) e a função *sscanf()* que lê uma *string.* Olhando atentamente para a função mencionada, vemos que ela é usada para pegar os dados de uma linha lida no arquivo do cuidador a ser analisado. Segue a função Atualizar Cuidador onde ela é encontrada:

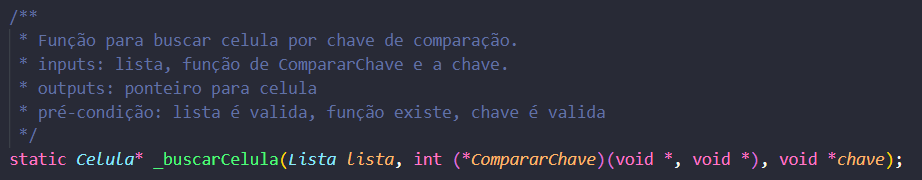


***TAD Lista***

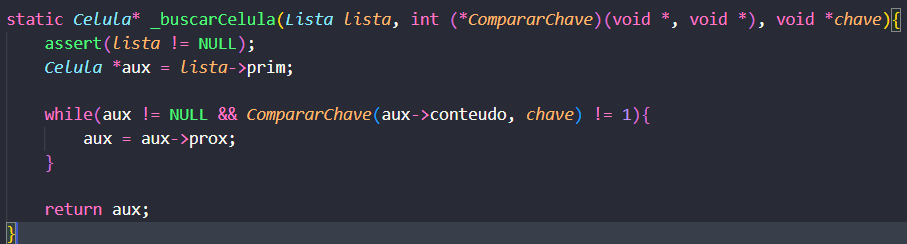
Agora estamos no *TAD* de Lista, aqui decidimos por implementar um tipo de lista que mais nos deixava confortável, logo escolhemos a lista duplamente encadeada com sentinela. Para isso usamos 2 *structs,* um de célula e um que implementa a sentinela da lista. Seguem eles abaixo:



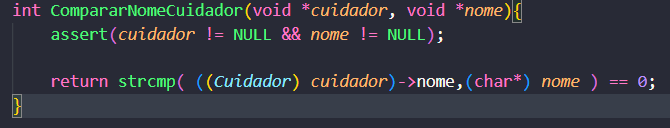
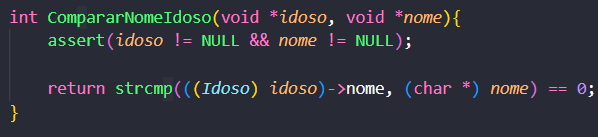
Como podemos ver na célula implementada temos um *void pointer* conteúdo para que essa lista possa ser utilizada em qualquer situação , com diferentes tipos (Cuidadores, Idosos, entre outros). Assim implementamos uma lista que é genérica. O resto das funções de lista são comuns para listas duplamente encadeadas como essa. Funções diferentes e para usos específicos também foi necessário a implementação, pois estamos usando uma lista genérica, um exemplo é a de buscar algo na lista:



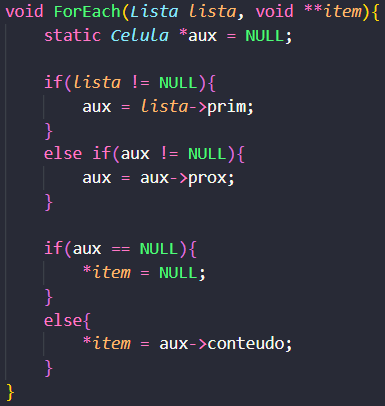
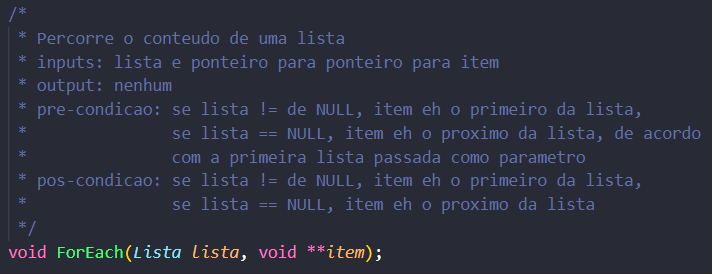
Para funcionar para todos os tipos, os próprios tipos devem ter sua função de comparar chaves de busca, para isso é necessário fazer uma função de comparar entre eles mesmo dentro de seus próprios *TADs,* que é passada como argumento na busca:



Exemplos abaixo dessas funções em Idoso.c e Cuidador.c respectivamente:

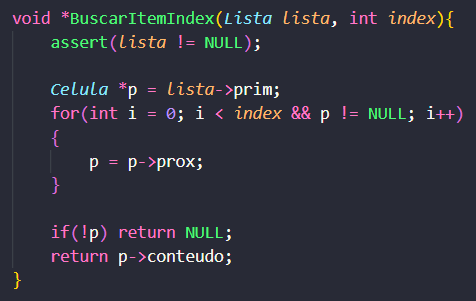
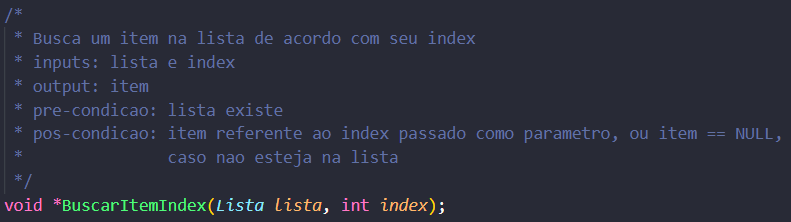


Outra função diferente é a *ForEach* ela é literalmente um for each que normalmente linguagens de programação orientadas a objetos tem.



Com ela podemos fazer um *for* menos complicado. Ela muda o conteúdo de *void\*\* item* que carrega o item da lista, isso tudo serve para manter a lista genérica. A função *ForEach()* é usada dentro de *EdCare.c* em um *for* para atualizar os dados dos Idosos e Cuidadores do sistema.

De função diferente temos também uma busca por *index,* que busca uma posição especifica da lista:



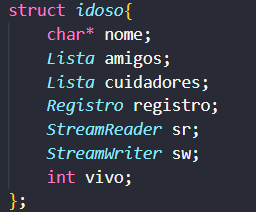
Essa função é utilizada em Desfazer Amizades no *TAD Idoso* quando algum Idoso é dado como falecido e no *TAD EdCare* para processar as informações e escrever a saída de cada Idoso.

Temos também a função de Deletar Lista que recebe uma função como parâmetro para desalocar também o espaço gastado pelo conteúdo das células da Lista:



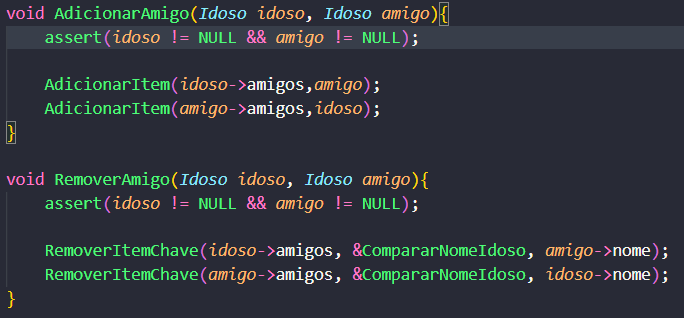
***TAD Idoso***

Para começarmos a falar desse TAD precisamos entender que cada Idoso carrega uma lista com Cuidadores e Amigos idosos, para chama-los caso algum caso de febre ou queda ocorra. Sendo assim o *struct* idoso fica dessa forma:

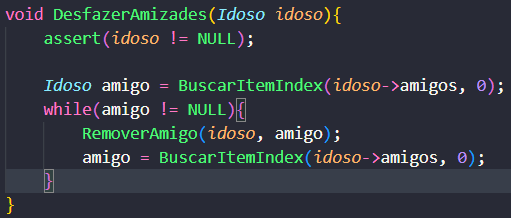


Aqui vemos o uso também de um *StreamWriter* para controlar as saídas do programa, que no caso depende dos dados recebidos para os Idosos.

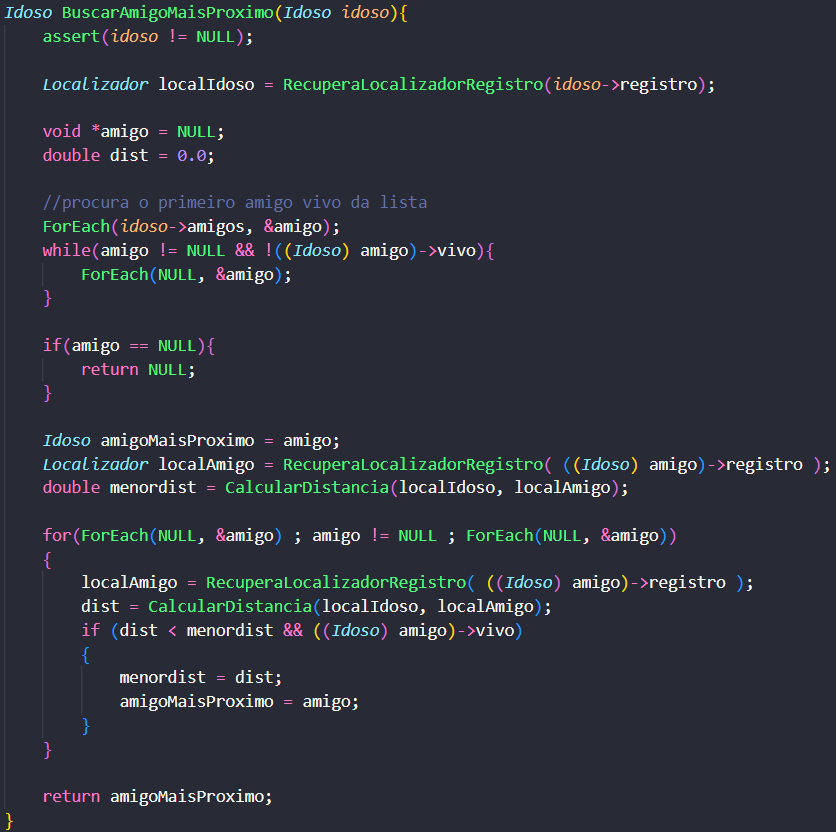
Nesse *TAD* temos o que controlar a lista de Amigos que nesse caso tem uma função para adicionar e remover amigos:



Aqui vemos o uso da função *CompararNomeIdoso*, que é passada como argumento na *RemoverItemChave* para que possamos fazer a busca corretamente para esse tipo Idoso. Temos uma função de Desfazer Amizades no caso do idoso falecer que chama Remover Amigo.

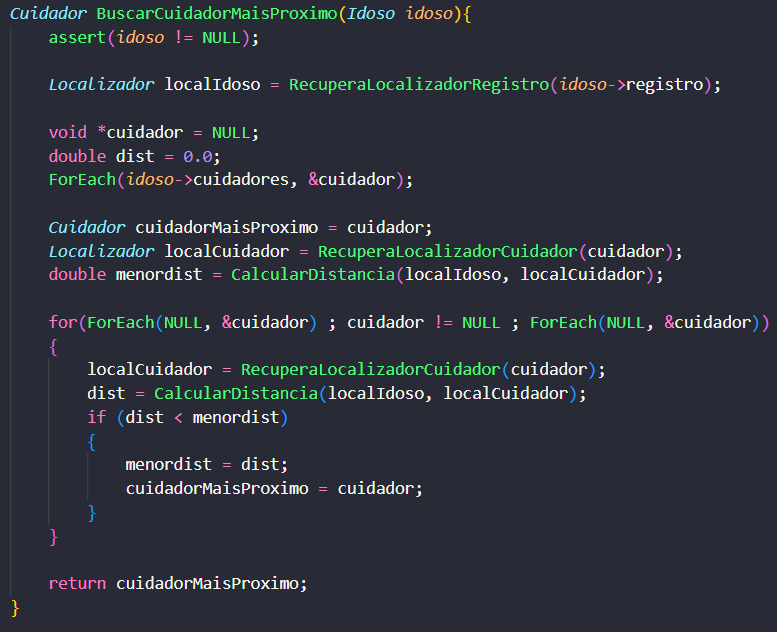
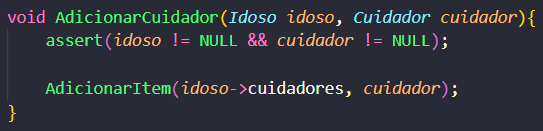


Para buscar o amigo mais próximo em caso de febre baixa temos uma função:



Ela busca na lista de amigos o mais próximo e usa função de *Localizador* para calcular as distancias .Podemos ver o uso do *ForEach* mais claramente aqui (quando passado *NULL* como argumento ele roda a lista assim não precisamos acessar células fora de funções de lista).

Agora para controlar a Lista de Cuidadores temos uma função de adicionar cuidadores e de chamar cuidador mais próximo.



A ultima usa também do *ForEach* e de funções de *Localizador* para fazer *cálculos.*

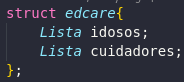
Agora chegamos na Atualiza Idoso que faz exatamente oque o nome diz.



Vemos que ela lê o arquivo de entrada do Idoso passado como parâmetro e puxa os dados dele com *sscanf(),* e manda para Atualiza Registro que muda o registro que fica guardado no Idoso.

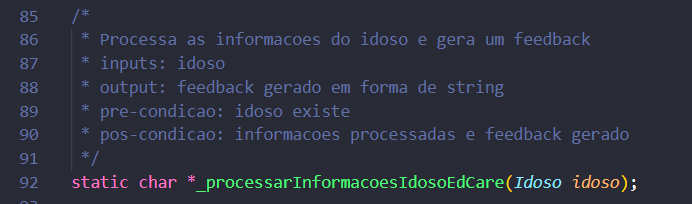
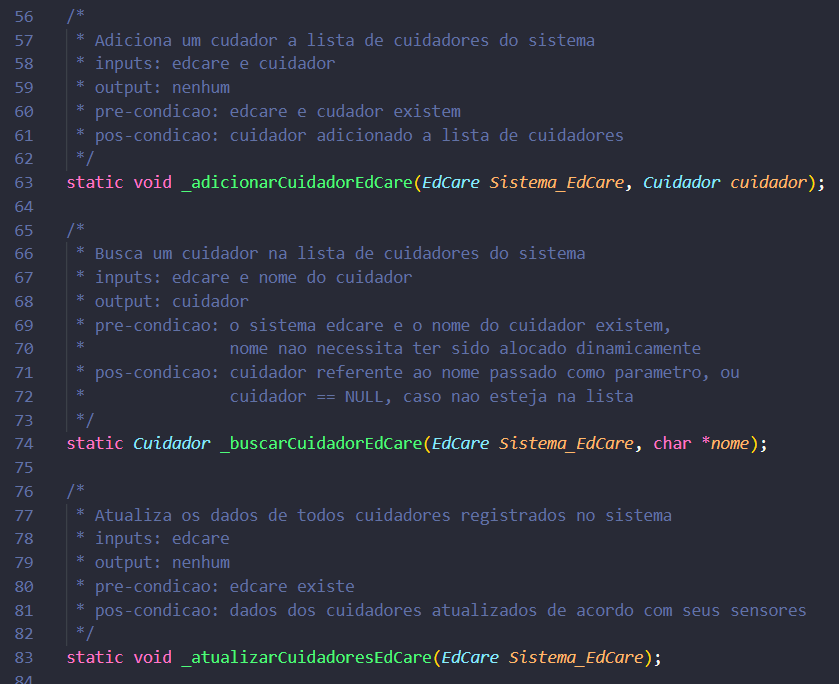
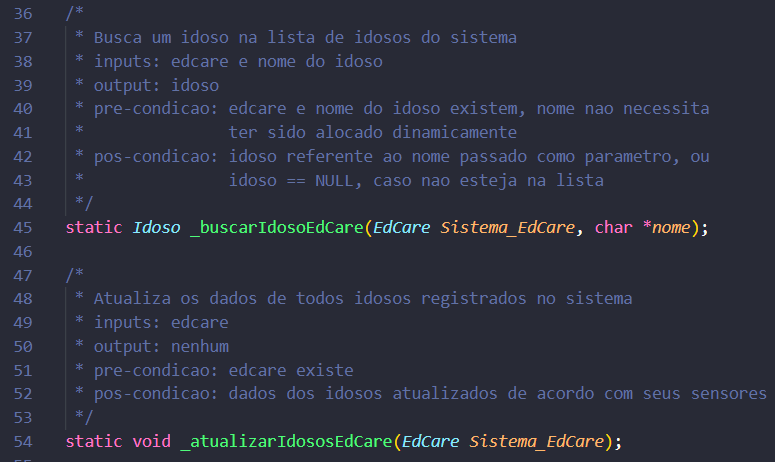
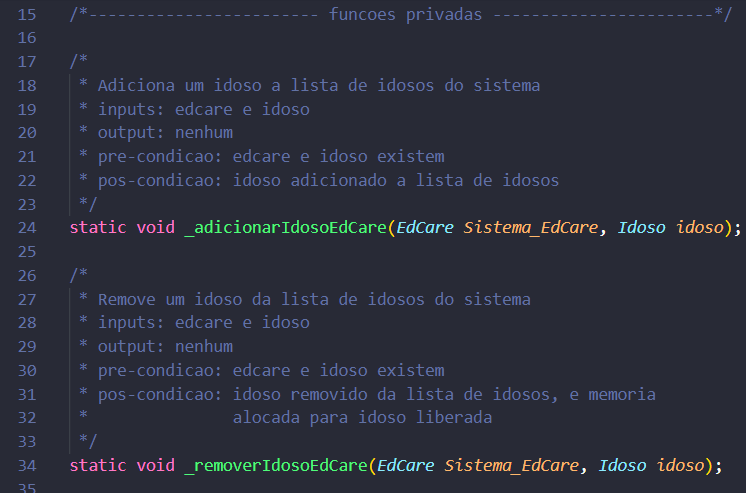
***TAD EdCare***

No *TAD EdCare* temos as funções de controle do sistema, com elas podemos atualizar o sistema a cada ciclo e analisar oque está acontecendo com os dados recebidos pelos arquivos de entrada.



Vemos que ele carrega uma lista de idosos e uma de cuidadores usando o mesmo tipo de lista, isso porque ela é genérica.

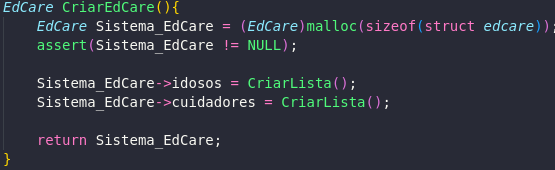
Nesse *TAD* temos o uso de diversas funções locais que somente são acessadas pelo próprio tipo. Seguem elas com suas explicações:



Vemos que algumas destas funções usam simplesmente de funções de outros tipos, que são de fácil compreensão e só estão aí para evitar acessos indevidos e *warnings*. As funções atualizar idosos, atualizar cuidadores e processar informações idoso serão explicadas mais a frente por serem mais complexas.

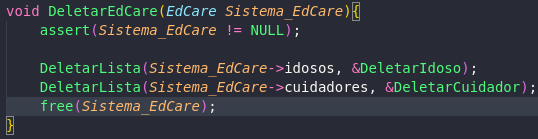
Na função de Criar temos simplesmente a alocação de memória e a criação das listas:

Lembrando aqui que usamos o tipo *EdCare* sendo um ponteiro para o *struct edcare .*

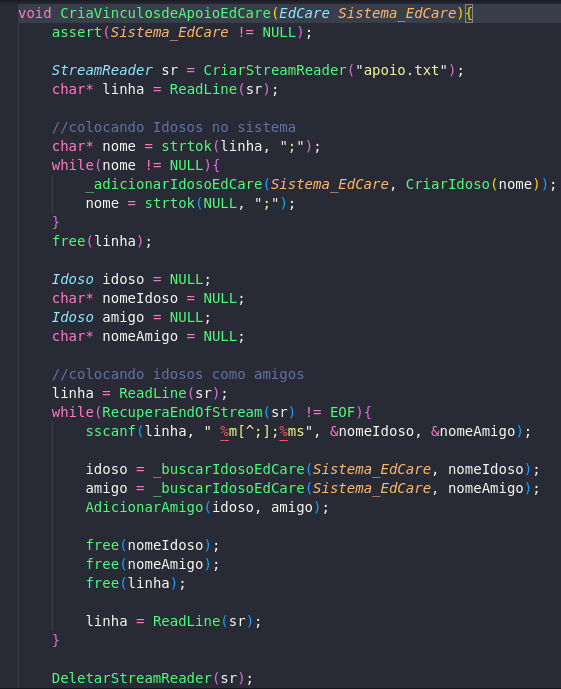


Seguindo para função de *Deletar EdCare*, que usa de funções da própria lista:

Vemos aqui o uso de funções deletar Idoso e cuidador que são passadas como parâmetro para a de deletar lista para apagar também os Idosos e os Cuidadores do sistema.



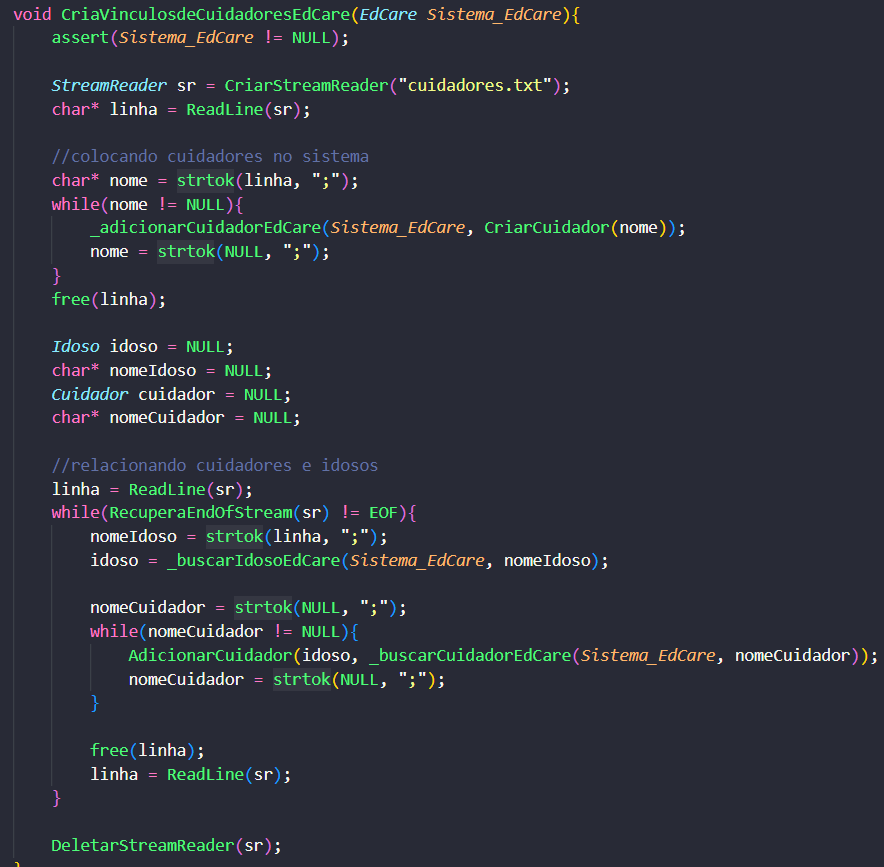
Agora iremos para uma função de inicialização do sistema, depois de ser criado, a função cria vínculos de apoio *EdCare*, que pega o arquivo *apoio.txt* e cria os laços estabelecidos dentro dele. Segue ela abaixo:



Essa função abre o arquivo *apoio.txt* , lê uma linha do aquivo e após pegar a linha ele usa a função *strtok()* para ler e dividir a *string* linha até o “*;*”, assim retirando somente os nomes dos idosos, para adiciona-los na lista do sistema. Observação, ela recebe *NULL* para continuar dividindo a mesma *string* usando *“;”* como referência.

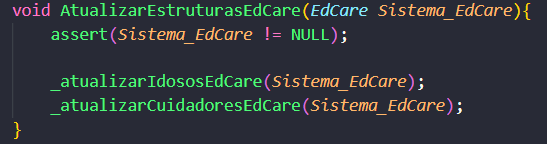
Depois a linha é desalocada, e partimos para a parte de fazer vínculos de amigos. Aqui usamos *TAG* no *scanf* para fazer um *malloc()* direto da *string* em *nomeIdoso e em nomeAmigo.* A *tag* usada é o *%m[^;]* que lê tudo até o “;” e o segundo *%ms* lê o resto da *string* toda, alocando diretamente seus respectivos tamanhos nas variáveis. Depois buscamos os Idosos na lista do sistema, para que possamos adicioná-los nas listas dos amigos,após ler o arquivo, utilizando a função *AdicionarAmigo*.

Agora a cria vinculos cuidadores que estabelece os laços que o arquivo *cuidadores.txt* estabelece:

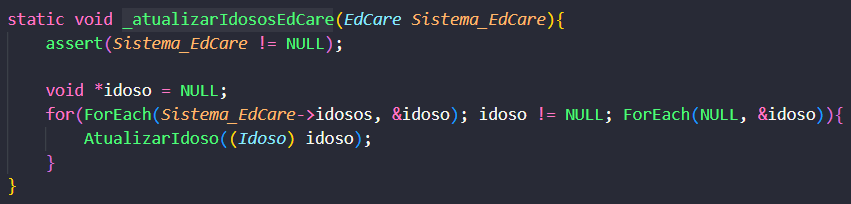


Aqui é utilizado a mesma ideia utilizada em vínculos de apoio, usando a função *strtok()* para ir buscando os cuidadores do sistema que estão inseridos no arquivo. O mesmo serve para colocar cuidadores nas listas dos idosos.

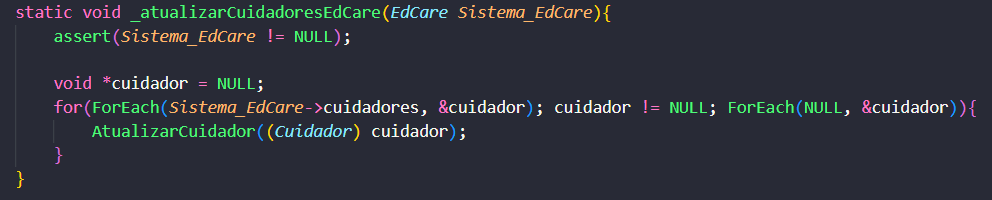
Agora uma função utilizada na main para atualizar o sistema a cada ciclo:



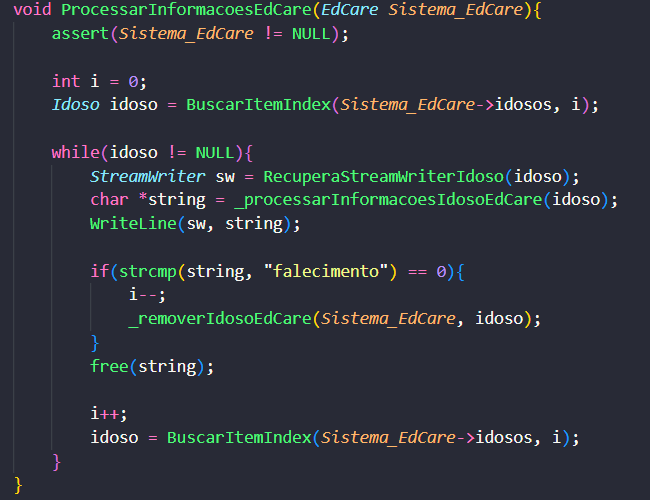
Vemos que é atualizado primeiro os idosos e depois cuidadores:



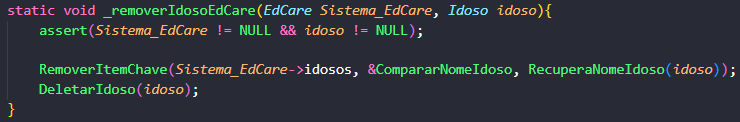
Usando funções de lista e dos próprios tipos fazemos as devidas alterações por cada ciclo.



Outra função chamada na main:



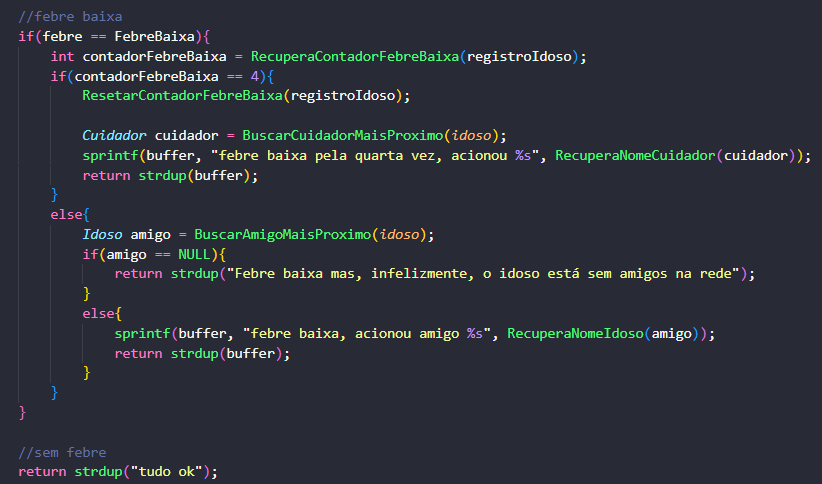
Com essa função acima buscamos os dados dos idosos que foram atualizados e gravamos as respectivas saídas que são escritas em processar informações idosos. Vemos que caso o idoso seja declarado falecido ele é removido do *EdCare,* segue abaixo a função que remove o idoso (usando função de lista e de idoso):



Lembrando que deletar Idoso também usa deletar amizades.



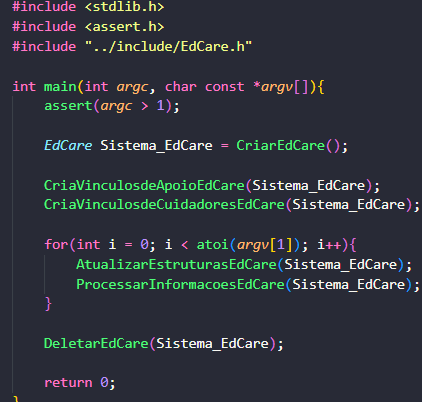
A função acima retorna oque será escrito no arquivo de saída do respectivo idoso. Caso esteja falecido ele já escreve na saída falecimento, caso aja queda ele procura o cuidador mais próximo e coloca quem foi acionado, caso aja 4 casos de febre baixa (ainda no caso de queda) o contador é *resetado,* verificamos também a febre (caso alta é *resetado* o contador de febres baixas). Caso pule e não tenha a queda vamos verificar estados de febre, caso alta ele chama o cuidador mais próximo e *reseta* contador de febre baixa, escreve na saída quem foi acionado.



No caso de febre baixa ele verifica o contador de febre baixa, caso == 4 chamamos o cuidador mais próximo, e escreve na saída. Caso contrário buscamos amigos mais próximo, e verificamos caso o idoso têm mesmo amigos. Sem febre escrevemos tudo ok. Lembrando que a função retorna a *string* que sera escrita usando *strdup()* para fazer isso, ou seja alocando espaço na *heap*, porém na função de processar informações do *EdCare* ela é desalocada.

***main.c***

Finalmente chegamos na *main*. Aqui simplesmente usamos funções de *EdCare,* para deixar bem intuitivo e de alto nível:



Observações: *assert* para verificar se a quantidade de ciclos foi passada na entrada padrão, a função *atoi()* serve para puxar o valor inteiro de *argv[1]* (o valor da quantidade de ciclos).

**Conclusão**

Após terminarmos o trabalho percebemos o quanto as listas são uteis para implementar esses tipos de sistemas e que com elas podemos juntar varias informações, vários tipos e organizar melhor nosso programa. De dificuldade, encontramos que a lista heterogênea é uma das mais complicadas de implementar, porém a mais útil caso o projeto necessite de varias listas que carreguem tipos diferentes.

**Bibliografia**

<https://www.tutorialspoint.com/c_standard_library/c_macro_assert.htm>

<https://www.tutorialspoint.com/c_standard_library/string_h.htm>

Materiais no classroom.