UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CAIO ALVES FIOROTTI
MATHEUS MEIER SCHREIBER

ESTRUTURAS DE DADOS

Trabalho 1

Vitória ES
2022

CAIO ALVES FIOROTTI
MATHEUS MEIER SCHREIBER

ESTRUTURAS DE DADOS 1

Trabalho 1

Documento apresentado ao Departamento de Engenharia da Computação da UFES, como requisito parcial para a conclusão do curso de Engenharia da Computação.
Orientador(a): Patricia Dockhorn Costa

Vitória ES
2022

SUMÁRIO

INTRODUÇÃO	**4**
IMPLEMENTAÇÃO	**5**
CONCLUSÃO	**14**
BIBLIOGRAFIA	**15**

1. INTRODUÇÃO

É fato que a população de idosos vem aumentando com o tempo e que precisamos cada vez mais direcionar cuidados especiais com a saúde e bem estar desses indivíduos. Dentro dessa perspectiva, o *EDCare* é um sistema que automatiza e facilita a mediação entre idosos e seus cuidadores. Obviamente, o presente documento será baseado em uma versão simplificada de um sistema passível de ser implementado dentro dos parâmetros reais de monitoramento corporal. No entanto, nada impede que as ideias aqui discutidas possam ser aprimoradas para comporem uma aplicação de sucesso no futuro.

Em poucas palavras, o programa prevê um sistema que, a partir de dados de entrada de um arquivo de texto, gere saídas em outro arquivo de texto. As respostas são baseadas no relacionamento entre posição, temperatura, sensor de queda, etc, de forma que tudo é atualizado a cada leitura das linhas dos arquivos.

O programa é feito em linguagem C utilizando, entre outras técnicas, listas encadeadas, ponteiros, alocação dinâmica de memória e tipos abstratos de dados.

2. IMPLEMENTAÇÃO

Inicialmente, a ideia foi criar quatro tipos abstratos de dados diferentes, são eles **Idoso**, **Cuidador**, que se referem aos indivíduos em específico, e também **Lista_I**, **Lista_C**, que armazenam as listas encadeadas de cada tipo. Os atributos presentes no tipo Idoso e Cuidador estão apresentados na Figura 1 visualizada abaixo.

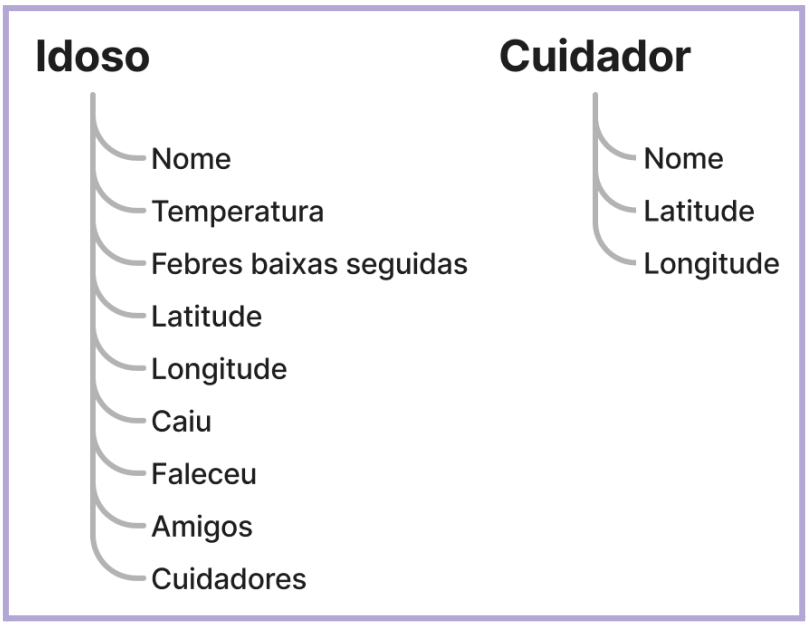

Figura 1 - Atributos do tipo Idoso e Cuidador

A escolha pelos dois tipos de listas diferentes se deve ao fato de cada uma conter tipos de células internas diferentes, onde uma aponta para o tipo Idoso e outra para o tipo Cuidador.

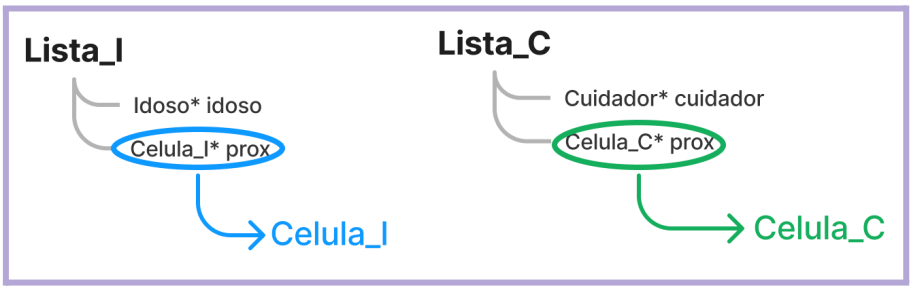

Figura 2 - Diferença entre os tipos de lista

Cada tipo foi implementado com suas **funções básicas** de inicialização, impressão, destruição, bem como **getters e setters** para ter acesso aos seus atributos particulares. Além disso, **outros métodos** foram utilizados ao longo da implementação, de forma que todos serão explicitados ao longo deste documento.

Dessa forma, a estruturação do sistema se baseia apenas em uma lista de idosos principal e uma lista de cuidadores. Um diagrama ilustrativo que demonstra de forma geral os alicerces do programa pode ser visualizado nas Figuras 3 e 4 a seguir.

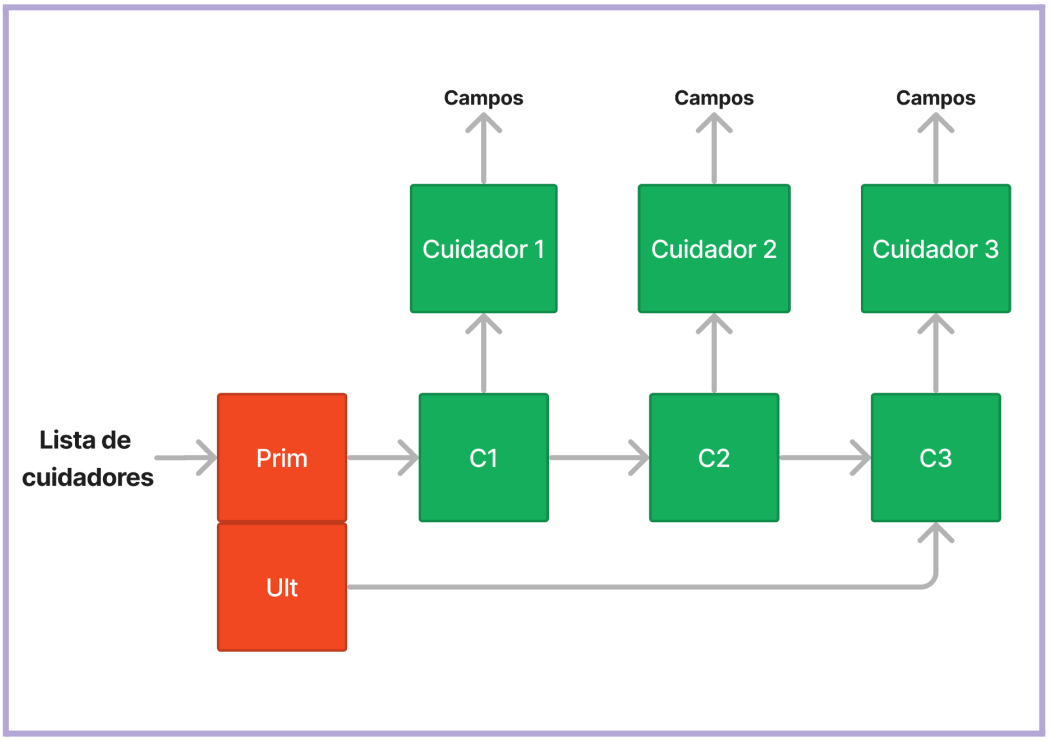

Figura 3 - Diagrama ilustrativo da lista de cuidadores

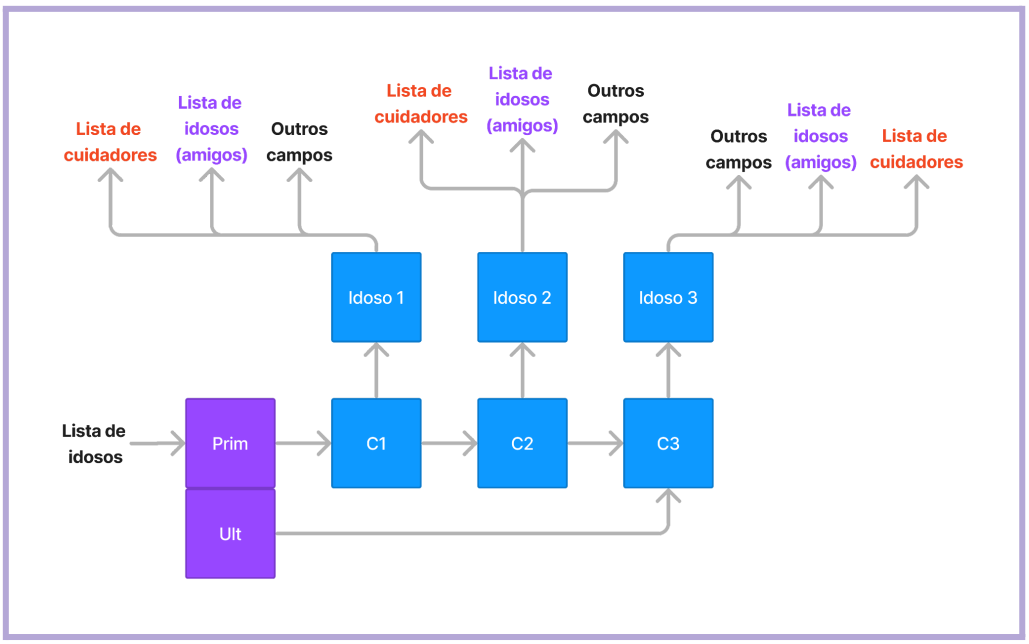

Figura 4 - Diagrama ilustrativo da lista de idosos

Uma vez analisada a estrutura geral do programa, o tipo de lista escolhido para a implementação foi a **lista simplesmente encadeada com sentinela**, já que a

sentinela auxilia na obtenção da posição dos elementos da lista e que não seria de nenhuma utilidade significativa realizar um duplo encadeamento. Além disso, essa escolha de lista pode ser atribuída ao fato de ser a de maior conforto e domínio entre os autores do projeto.

Com esse contexto em mente, parte-se para a análise das funções e da execução do programa. Primeiramente, é importante ressaltar que a execução foi dividida em dois momentos importantes: a **leitura** dos dados e o **registro** do resultado das relações entre esses dados nos arquivos de saída, sendo que cada um dos momentos engloba vários métodos e funcionalidades.

Dessa forma, a porção inicial do arquivo testador (*main.c)* é voltada para a leitura dos nomes dos indivíduos e das relações interpessoais entre idosos e cuidadores, como quem é amigo de quem, e também quem é cuidador de quem.

Em seguida, pode-se iniciar as análises da **função principal** do programa, a **`FazLeituraIdosos`**, a qual é responsável por de fato fazer a leitura dos dados sensoriais nos arquivos dos idosos (e seus cuidadores) e depois imprimir a saída esperada em outros arquivos. Sua implementação está exposta nas Figuras 5 e 6.

```c
void FazLeituraIdosos(Lista_I* listaIdosos, int qtdLeituras) {

    char diretorio[100], diretorio2[100];
    int i = 0, j = 0;

    Celula_I* p;
    Idoso* idosoASerMatado = NULL;

    //limpado todos os arquivos de saida dos idosos, e criando novos caso já nao existam
    for(p=listaIdosos->Prim;p!=NULL;p=p->prox) {
        sprintf(diretorio, "saidas/%s-saida.txt", getNomeIdoso(p->idoso));
        FILE* limpaArquivo = fopen(diretorio, "w");
        fclose(limpaArquivo);
    }

    while (i < qtdLeituras) {

        int caiu=0, faleceu = 0;
        double temp = 0.0, lat = 0.0, longi = 0.0;
        p = listaIdosos->Prim;

        while (p != NULL) {

            /* abrindo o arquivo de entrada do idoso */
            sprintf(diretorio, "entradas/%s.txt", getNomeIdoso(p->idoso));
            FILE* arquivoEntrada = fopen(diretorio, "r");

            // atualiza a localização dos cuidadores do idoso que esta sendo lido
            AtualizaDadosCuidadores(getCuidadoresIdoso(p->idoso), i+1);


            // caso nao seja a primeira leva de leituras, então deve-se desconsiderar todas as linhas
            // do arquivo que ja foram lidas nos loops passados
            if (i>0) {
                for(j=0;j<i;j++) fscanf(arquivoEntrada, "%*[^\n]\n");

                if (j!=i) {
                    printf("ERRO COM: %s\n", getNomeIdoso(p->idoso));
                    continue;
                }
            }
```

Figura 5 - Parte 1 da implementação da função **`FazLeituraIdosos`** (arquivo *Lista_I.c*)

```
            }

            /* se ler a temperatuda (primeira leitura) com erro na verdade é o falecimento */
            if (fscanf(arquivoEntrada, "%lf;", &temp) != 1) {

                idosoASerMatado = p->idoso;
                setFaleceuIdoso(p->idoso, 1);
                faleceu = 1;

            } else {

                setTempIdoso(p->idoso, temp);

                //febre baixa
                if (getTempIdoso(p->idoso)>=37 && getTempIdoso(p->idoso)<38) {
                    setFebreSeguidasIdoso(p->idoso, getFebreSeguidasIdoso(p->idoso)+1);
                }

                fscanf(arquivoEntrada, "%lf;", &lat);
                setLatIdoso(p->idoso, lat);

                fscanf(arquivoEntrada, "%lf;", &longi);
                setLongiIdoso(p->idoso, longi);

                fscanf(arquivoEntrada, "%d", &caiu);
                setCaiuIdoso(p->idoso, caiu);


            }

            fclose(arquivoEntrada);
            p = p->prox;
        }

        RegistraInfosIdosos(listaIdosos);
        i++;

    }
}
```

Figura 6 - Parte 2 da implementação da função **`FazLeituraIdosos`** (arquivo *Lista_I.c*)

A **`FazLeituraIdosos`** é construída de forma a seguir os seguintes objetivos: **garantir arquivos limpos** para receber as saídas da execução, **atualizar os dados** dos cuidadores e dos idosos, e por fim realizar a lógica que envolve as ações e **decisões** a se tomar a cada leitura. Em seu início, consta um loop simples para abrir os arquivos de saída de todos os idosos, criando novos se necessário, no intuito de manter tudo limpo e preparado para receber novas informações.

Em seguida, são previstos ***loops* encadeados**, seguindo a seguinte configuração: um loop principal pelo **número de leituras** especificado na chamada da aplicação e outro loop interno pelo **número de idosos** presentes na lista de idosos previamente estruturada. O loop externo cuida de iterar por cada uma das leituras em todos os arquivos de entrada dos idosos, e registrar as respostas adequadas nos arquivos de saída. Enquanto isso, o loop interno cuida de ler cada arquivo de entrada e apenas

armazenar essas leituras nos atributos dos tipos Idoso e Cuidador. A representação esquemática desse encadeamento pode ser observado na Figura 7 abaixo.

Figura 7 - Diagrama ilustrativo dos loops encadeados e seus intervalos de ação.

No início do loop mais interno, como se está passando idoso por idoso, criou-se uma função para **atualizar os dados posicionais** de todos os cuidadores referentes ao idoso atual do loop, sendo que essa função cuida da abertura e fechamento do arquivo de cada Cuidador. Em sequência, é realizada uma **sincronização** entre a linha que está sendo lida do arquivo do idoso, e o número da leitura atual. Isso se dá devido ao fato de que como os arquivos dos idosos são abertos e fechados várias vezes durante o loop exterior, a cada leitura é necessário ajustar o buffer para que a linha que está sendo lida no arquivo do idoso não esteja sendo lida mais de uma vez, criando valores duplicados. O diagrama descrito na Figura 8 a seguir busca ilustrar essa tratativa.

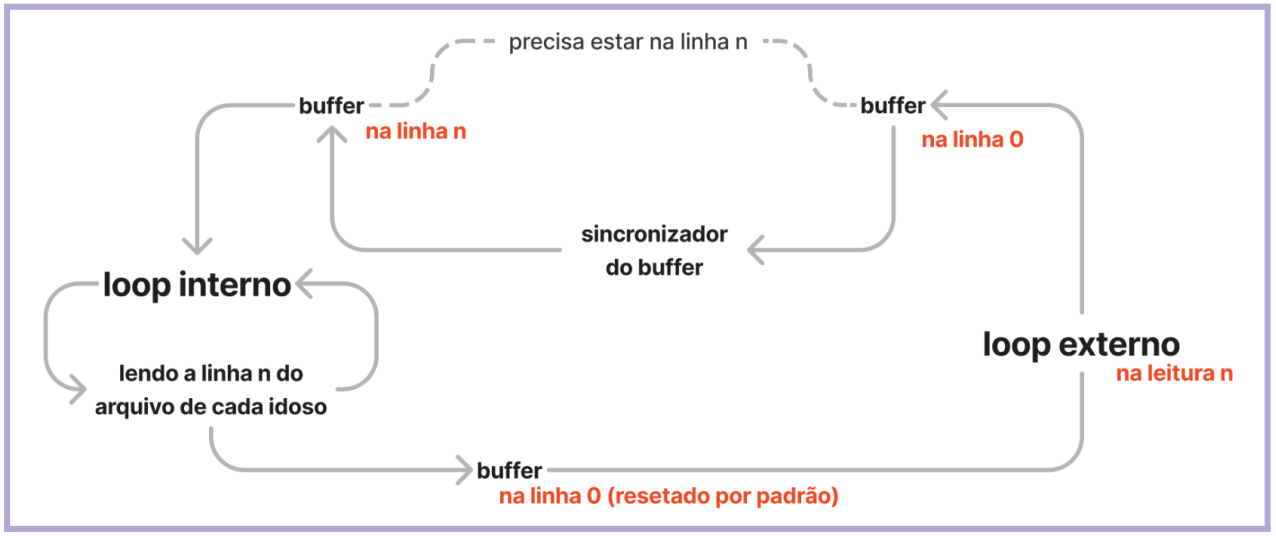

Figura 8 - Diagrama representativo da lógica por trás da sincronização do buffer.

Em seguida, ainda no loop interno, é feito um teste para verificar se a linha atual lida do arquivo do idoso é referente ao **falecimento** do mesmo. Esse teste é feito por um

scanf que tenta ler um número ***float*** (temperatura do idoso), e caso não consiga e tenha lido um caractere, quer dizer que, ao invés da temperatura, foi passado o estado de falecimento. Toda essa tratativa gira em torno do fato que, por padrão, a temperatura é passada como primeiro parâmetro em todos os casos de não-falecimento.

Dessa forma, caso a temperatura tenha sido lida, parte-se para armazenar as seguintes informações nos atributos do idoso atual do loop: **latitude**, **longitude** e valor *booleano* para **queda**. Já nessa etapa, mesmo que ainda estamos apenas nas leituras dos valores, é elaborada uma lógica rápida para detectar se houve febre baixa, para que o atributo de "*febres baixas seguidas*" seja atualizado antes de se passar para a lógica principal que gerencia o registro das saídas.

Após realizadas as leituras e armazenadas nos atributos dos indivíduos correspondentes, pode-se partir para processar as respostas a essas leituras e registrá-las nos devidos arquivos, sendo que isso é feito ainda no loop externo, logo após o fim do loop interno. A função responsável por todo esse processamento é a `RegistraInfosIdosos`, que pode ser visualizada nas Figuras 9 e 10 abaixo.

```
190  void RegistraInfosIdosos(Lista_I* listaIdosos){
191
192      char diretorio[100];
193      Celula_I* p = NULL;
194      Celula_I* aux = NULL;
195
196      p = listaIdosos->Prim;
197
198      while(p!=NULL){
199
200          sprintf(diretorio, "saidas/%s-saida.txt", getNomeIdoso(p->idoso));
201          FILE* arquivoSaida = fopen(diretorio, "a");
202
203          //falecimento
204          if (getFaleceuIdoso(p->idoso)) {
205
206              fprintf(arquivoSaida, "falecimento\n");
207
208              aux=p->prox;
209              MataIdoso(p->idoso, listaIdosos);
210              p=aux;
211              fclose(arquivoSaida);
212              continue;
213
214          //queda
215          } else if (getCaiuIdoso(p->idoso)) {
216
217              fprintf(arquivoSaida, "queda");
218              AcionaCuidadorMaisProximo(getLatIdoso(p->idoso), getLongiIdoso(p->idoso), getCuidadoresIdoso(p->idoso),
219                  arquivoSaida);
220
221              setCaiuIdoso(p->idoso, 0);
222              if (getTempIdoso(p->idoso)>=38) setFebreSeguidasIdoso(p->idoso, 0);
223
224          //febre alta
225          } else if (getTempIdoso(p->idoso)>=38) {
```

Figura 9 - Trecho do código da **`RegistraInfosIdosos`** (Lista_I.c) responsável pelo registro das informações nos arquivos.

```
        //febre alta
        } else if (getTempIdoso(p->idoso)>=38) {
            fprintf(arquivoSaida, "febre alta");
            AcionaCuidadorMaisProximo(getLatIdoso(p->idoso), getLongiIdoso(p->idoso), getCuidadoresIdoso(p->idoso),
             arquivoSaida);
            setFebreSeguidasIdoso(p->idoso, 0);
        } else {

            //febre baixa
            if (getTempIdoso(p->idoso)>=37 && getTempIdoso(p->idoso)<38) {

                //4 febres baixas seguidas
                if (getFebreSeguidasIdoso(p->idoso)>=4) {

                    fprintf(arquivoSaida, "febre baixa pela quarta vez");
                    AcionaCuidadorMaisProximo(getLatIdoso(p->idoso), getLongiIdoso(p->idoso), getCuidadoresIdoso(p->idoso),
                     arquivoSaida);
                    setFebreSeguidasIdoso(p->idoso, 0);

                } else{

                    fprintf(arquivoSaida, "febre baixa");
                    AcionaAmigoMaisProximo(getLatIdoso(p->idoso), getLongiIdoso(p->idoso), getAmigosIdoso(p->idoso),
                     arquivoSaida);

                }

            //tudo ok
            } else {
                fprintf(arquivoSaida, "tudo ok\n");
            }
        }
        fclose(arquivoSaida);
        p=p->prox;
    }
}
```

Figura 10 - Trecho do código da **`RegistraInfosIdosos`** (Lista_I.c) responsável pelo registro das informações nos arquivos.

Com isso, a **`RegistraInfosIdosos`** serve para relacionar os atributos atuais dos idosos, com as saídas possíveis. Nesse sentido, por meio de um loop pela lista de idosos, primeiramente é verificado se o idoso faleceu, pelo valor booleano no tipo Idoso. Em caso verdadeiro, é registrado o falecimento no arquivo de saída, e retira-se esse idoso da trama relacional de amizades de todos os outros idosos, por meio da comparação um a um se cada um dos idosos tinha um vínculo com o falecido. O tratamento do estado de falecido é representado logo abaixo, pela Figura 11.

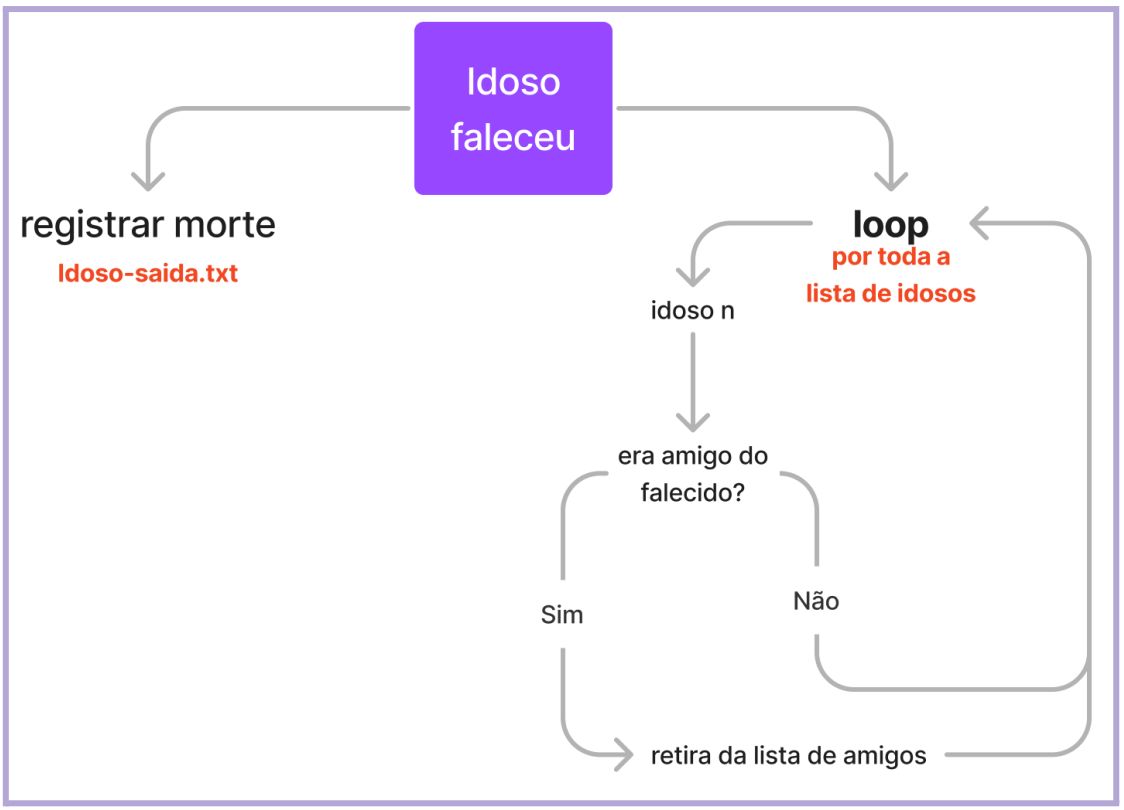

Figura 11 - Diagrama representativo da lógica envolvendo o falecimento de um idoso.

Caso isso não seja verdade, é verificado se o idoso caiu, de forma que é acionado o cuidador mais próximo em caso verdadeiro. Esse acionamento é feito por meio de uma função externa ao TAD **Lista_I**, compreendida no TAD **Lista_C**, que faz o cálculo euclidiano da distância entre o ponto descrito pela latitude e longitude do cuidador e o ponto descrito pela latitude e longitude do idoso. A lógica por trás desse cálculo e sua aplicação na função pode ser observada na Figura 12 a seguir.

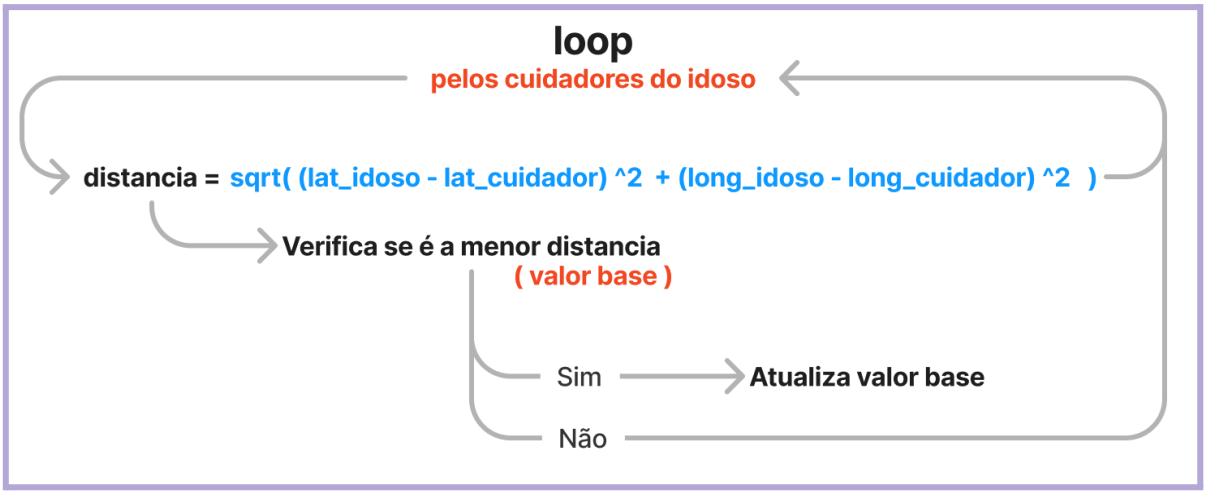

Figura 12 - Diagrama que ilustra o uso cálculo euclidiano de distância no acionamento de cuidadores.

Em seguida, é verificada a temperatura do idoso, para saber se o mesmo está com febre alta, ou seja, maior ou igual a 38 graus, para se acionar um cuidador em caso verdadeiro. Caso seja falso e a temperatura ainda esteja acima de 37 graus ou equivalente, é acionado o amigo mais próximo, considerando que o idoso tenha amigos, senão ninguém é chamado. Além disso, é feita outra verificação, para caso o idoso já esteja na quarta febre baixa seguida, acionando um cuidador caso verdadeiro. Toda a tratativa que engloba a temperatura do idoso é ilustrada na Figura 13 abaixo.

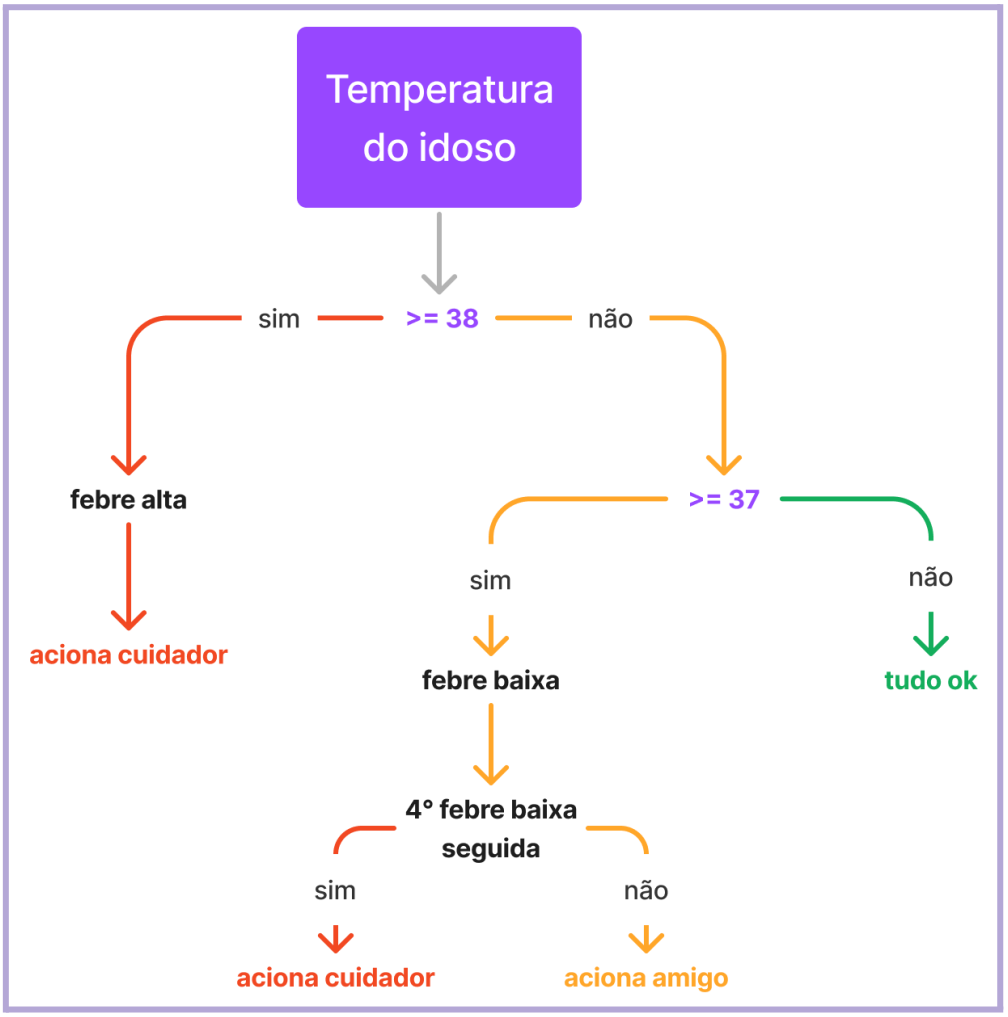

Figura 13 - Diagrama que ilustra a lógica envolvendo a temperatura do idoso.

Nesse contexto, é registrado o estado “tudo ok”, caso o idoso não tenha apresentado nenhum dos problemas citados anteriormente, partindo-se para a próxima iteração do loop externo da função `FazLeituraIdosos`, repetindo todo esse procedimento para cada uma das linhas de dados sensoriais.

Por fim, com a leitura e registro da última linha de acordo com a quantidade especificada na chamada do programa realizados, é feita a **liberação** de toda a memória alocada envolvendo as duas listas principais (idosos e cuidadores), e então encerra-se o programa.

3. CONCLUSÃO

Na implementação escolhida, decidiu-se ler os dados dos arquivos de texto sem funções de bibliotecas específicas, mas sim via caracter a caracter. Dessa forma tem-se um maior controle do que está sendo lido e do que se quer ler. Porém, essa iniciativa gerou alguns problemas devido aos caracteres em branco (*whitespaces)* que estavam presentes em alguns dos casos de teste. Isso porque as leituras não eram finalizadas e forneciam nomes “errados”.

Outro empecilho foi redigir o atual documento, que foi deixado apenas ao final da construção do código, ao invés de ir documentando ao longo da edificação do trabalho. Inicialmente tal ideia tinha sentido, já que não se sabia de fato quais métodos e algoritmos de fato iriam estar presentes no final, porém ao passar do tempo, quando o código já estava direcionado, o ideal seria fazer a documentação em conjunto com o código de fato, já que, inclusive, auxiliaria no próprio entendimento do trabalho.

A escolha de listas simplesmente encadeadas com sentinela não apresentou um problema, já que era a matéria abordada mais recentemente nas aulas do curso, logo os autores tiveram mais contato até o momento, portanto mais facilidade. O encadeamento simples da pilha não foi uma adversidade, uma vez que “andar” nas duas direções da lista não foi necessário.

Por fim, acreditamos que esse trabalho auxiliou muito no entendimento da matéria, especialmente no sentido de ser aplicado em um sistema mais próximo da realidade, incentivando, dessa forma, o raciocínio para a resolução de problemas.

4. BIBLIOGRAFIA

Vídeos-aula de Lista Simplesmente Encadeada com Sentinela. Classroom de Estrutura de Dados I, prof Patrícia Dockhorn Costa.

TABELA ASCII: American Standard Code for Information Interchange. [*S. l.*], 16 jun. 2015. Disponível em: https://web.fe.up.pt/~ee96100/projecto/Tabela%20ascii.htm. Acesso em: 28 maio 2022.