Fase 1 - Listas ligadas

Gerado por Doxygen 1.10.0

Capítulo 1

Estruturas de Dados Avançadas (EDA) - Fase 1

1.1 Instituto Politécnico do Cávado e do Ave (IPCA) - Barcelos

1.1.1 Aluno: Hugo Cruz a23010

1.1.2 Motivação

Este projeto individual tem como objetivo reforçar e aplicar os conhecimentos adquiridos na Unidade Curricular de Estruturas de Dados Avançadas (EDA), especialmente no que diz respeito à manipulação de estruturas de dados dinâmicas na linguagem de programação C.

1.1.3 Objetivo

Desenvolver soluções de software para manipulação de listas ligadas representando matrizes de inteiros, aplicando os conhecimentos adquiridos sobre estruturas de dados dinâmicas, armazenamento em ficheiro, modularização e documentação com Doxygen.

1.1.4 Fase 1 - Listas Ligadas

1.1.4.1 Descrição do Problema

Implementar uma solução para calcular a soma máxima possível dos inteiros de uma matriz, de modo que nenhum dos inteiros selecionados compartilhe a mesma linha ou coluna.

1.1.4.2 Funcionalidades a Implementar

1. **Estrutura de Dados Dinâmica**: Utilizar listas ligadas para representar a matriz de inteiros.
2. **Carregamento de Dados**: Carregar a matriz a partir de um ficheiro de texto, considerando qualquer dimensão, com os valores separados por vírgulas.
3. **Alteração de Valores**: Permitir a alteração dos inteiros na estrutura de dados.
4. **Inserção de Linhas/Colunas**: Adicionar novas linhas ou colunas na matriz.
5. **Remoção de Linhas/Colunas**: Eliminar linhas ou colunas existentes.
6. **Listagem Tabular**: Exibir todos os inteiros da matriz de forma tabular na consola.
7. **Cálculo da Soma Máxima**: Determinar a soma máxima possível dos inteiros, respeitando a condição de exclusividade de linha e coluna.

1.1.5 Documentação

- **Relatório Doxygen:** doc/latex/Relatório-Latex.pdf
- **Código Fonte:** src/

Capítulo 2

Índice das estruturas de dados

2.1 Estruturas de dados

Lista das estruturas de dados com uma breve descrição:

Capítulo 3

Índice dos ficheiros

3.1 Lista de ficheiros

Lista de todos os ficheiros com uma breve descrição:

Capítulo 4

Documentação da estruturas de dados

4.1 ReferÃªncia Ã estrutura Lista

`#include <header.h>`

Campos de Dados

- int num
- struct Lista * prox

4.1.1 DocumentaÃ§Ã£o dos campos e atributos

4.1.1.1 num

`int num`
Variavel que armazena os numeros

4.1.1.2 prox

`struct Lista * prox`
Apontador para o proximo número

A documentaÃ§Ã£o para esta estrutura foi gerada a partir dos seguintes ficheiros:

- C:/Users/hugoc/OneDrive - Instituto PolitÃ©cnico do CÃ¡vado e do Ave/2023_2024/Estruturas de Dados AvanÃ§adas/Dev/Listas ligadas - Fase 1/src/FunÃ§Ãµes/header.h
- C:/Users/hugoc/OneDrive - Instituto PolitÃ©cnico do CÃ¡vado e do Ave/2023_2024/Estruturas de Dados AvanÃ§adas/Dev/Listas ligadas - Fase 1/src/ResoluÃ§Ã£o/Lib/header.h

4.2 Referência à estrutura Matriz

`#include <header.h>`

Campos de Dados

- Lista * linha
- struct Matriz * ProxLinha

4.2.1 Documentação dos campos e atributos

4.2.1.1 linha

`Lista * linha`
Apontador para listas

4.2.1.2 ProxLinha

```
struct Matriz * ProxLinha
```

Apontador para a proxima lista

A documentação para esta estrutura foi gerada a partir dos seguintes ficheiros:

- C:/Users/hugoc/OneDrive - Instituto Politécnico do Cávado e do Ave/2023_2024/Estruturas de Dados Avançadas/Dev/Listas ligadas - Fase 1/src/Funções/header.h
- C:/Users/hugoc/OneDrive - Instituto Politécnico do Cávado e do Ave/2023_2024/Estruturas de Dados Avançadas/Dev/Listas ligadas - Fase 1/src/Resolução/Lib/header.h

Capítulo 5

Documentação do ficheiro

5.1 Referência ao ficheiro C:/Users/hugoc/OneDrive - Instituto Politécnico do Cávado e do Ave/2023_2024/Estruturas de Dados Avançadas/Dev/Listas ligadas - Fase 1/README.md

5.2 ReferÃªncia ao ficheiro C:/Users/hugoc/OneDrive - Instituto PolitÃ©cnico do CÃ¡vado e do Ave/2023_2024/Estruturas de Dados AvanÃ§adas/Dev/Listas ligadas - Fase 1/src/FunÃ§Ãµes/header.h

```
#include <stdbool.h>
```

Estruturas de Dados

- struct Lista
- struct Matriz

DefiniÃ§Ãµes de tipos

- typedef struct Lista Lista
- typedef struct Matriz Matriz

FunÃ§Ãµes

- int * CarregaDados (int *tamanho)

  Carrega dados de um ficheiro CSV.
- Lista * CriarNumero (int n)

  Cria memória para alocar a estrutura lista.
- Lista * ColocarNaLista (Lista *inicio, int n)

  Insere um número no final da lista.
- Matriz * CriarMatriz (Lista *lista)

  Cria uma Matriz.
- Matriz * ColocarNaMatriz (Matriz *inicio, Lista *linha)

  Insere uma nova linha na matriz.
- Matriz * DistribuirDados (int linhas, int colunas)

  Distribui os dados lidos de um arquivo em uma matriz dinâmica.
- void MostrarMatriz (Matriz *m)

  Mostra os elementos de uma matriz no console.
- bool MudarValor (Matriz *matriz, int linha, int coluna, int ValorMudar)

  Modifica o valor de um elemento específico na matriz.

- Matriz ∗ InserirLinha (Matriz ∗matriz, int linha, int tamanhoC, int ∗valores)

  Insere uma nova linha após uma linha específica na matriz.
- Matriz ∗ InserirColuna (Matriz ∗matriz, int posicao, int ∗valores)

  Insere uma nova coluna em uma posição específica em todas as linhas da matriz.
- Matriz ∗ RemoverLinha (Matriz ∗matriz, int linha)

  Remove uma linha específica da matriz.
- Matriz ∗ RemoverColuna (Matriz ∗matriz, int coluna)

  Remove uma coluna específica de todas as linhas da matriz.
- int somaMaiores (Matriz ∗m)

  Soma os valores maiores da linha.

5.2.1 DocumentaÃ§Ã£o dos tipos

5.2.1.1 Lista

```
typedef struct Lista Lista
```

5.2.1.2 Matriz

```
typedef struct Matriz Matriz
```

5.2.2 DocumentaÃ§Ã£o das funÃ§Ãµes

5.2.2.1 CarregaDados()

```
int * CarregaDados (
            int * tamanho )
```

Carrega dados de um ficheiro CSV.

ParÂ¢metros

tamanho	Guarda quantos dados foram lidos

Retorna

int∗ Apontador que retorna os dados lidos.

```
00014                                   {
00015     int i = 0;
00016     int* aux = (int*)malloc(sizeof(int) * 50); // Aloca memória para 50 inteiros.
00017
00018     FILE* ficheiro = fopen("matriz.csv", "r"); // Abrir o arquivo.
00019
00020     if (ficheiro == NULL) return NULL; // Retorna NULL caso o ficheiro não seja aberto.
00021
00022     // Lê os dados separados por ponto e virgula enquanto houver dados.
00023     while (fscanf_s(ficheiro, "%d;", &aux[i]) == 1)
00024     {
00025         i++;
00026     }
00027
00028     *tamanho = i;     // Armazena a quantidade de numeros lidos.
00029     fclose(ficheiro); // Fecha o arquivo.
00030
00031     return aux; // Retorna um apontador para os inteiros armazenados.
00032     free(aux);
00033 }
```

5.2.2.2 ColocarNaLista()

```
Lista * ColocarNaLista (
            Lista * inicio,
            int n )
```

Insere um número no final da lista.

ParÃ¢metros

inicio	Apontador para o primeiro elemento da lista.
n	O número a ser inserido na lista.

```
00062 {
00063     Lista* nova = CriarNumero(n); // Cria um novo elemento.
00064     if (nova == NULL)
00065         return inicio; // Retorna o início.
00066 
00067     if (inicio == NULL)
00068     {
00069         inicio = nova; // Se a lista está vazia, o novo elemento é colocado no início.
00070     }
00071 
00072     else
00073     {
00074         Lista* aux2 = inicio;
00075         // Percorre a lista até o final.
00076         while (aux2->prox != NULL)
00077         {
00078             aux2 = aux2->prox;
00079         }
00080         aux2->prox = nova; // Insere o novo elemento no final da lista.
00081     }
00082 
00083     return inicio; // Retorna um apontador para o primeiro elemento da lista.
00084 }
```

5.2.2.3 ColocarNaMatriz()

```
Matriz * ColocarNaMatriz (
            Matriz * inicio,
            Lista * linha )
```

Insere uma nova linha na matriz.

ParÃ¢metros

inicio	Apontador para a primeira linha da matriz.
linha	Apontador para a lista que será inserida como nova linha.

```
00112 {
00113 
00114     Matriz* nova = CriarMatriz(linha); // Cria uma nova linha com a lista fornecida.
00115 
00116     if (nova == NULL) return inicio; // Retorna o início se falha ao criar a nova linha.
00117 
00118     if (inicio == NULL)
00119     {
00120         inicio = nova; // Se a matriz está vazia, a nova linha é o início.
00121     }
00122 
00123     else
00124     {
00125         Matriz* aux2 = inicio;
00126 
00127         // Percorre a matriz até a última linha.
00128         while (aux2->ProxLinha != NULL)
00129         {
00130             aux2 = aux2->ProxLinha;
00131         }
00132         aux2->ProxLinha = nova; // Insere a nova linha no final da matriz.
00133     }
00134     return inicio; // Retorna um apontador para a primeira linha da matriz.
00135 }
```

5.2.2.4 CriarMatriz()

```
Matriz * CriarMatriz (
            Lista * lista )
```

Cria uma Matriz.

ParÃ¢metros

lista	Apontador para a lista que é uma linha da matriz.

```
00091                                   {
00092
00093     Matriz* aux = (Matriz*)malloc(sizeof(Matriz) * 1); // Aloca memória para uma nova linha da matriz.
00094
00095     if (aux != NULL)
00096     {
00097         aux->linha = lista;    // Define a lista como linha da matriz.
00098         aux->ProxLinha = NULL; // Define a próxima linha vazia
00099     }
00100
00101     return aux; // Retorna um apontador para a nova linha da matriz.
00102     free(aux);
00103 }
```

5.2.2.5 CriarNumero()

```
Lista * CriarNumero (
            int n )
```

Cria memória para alocar a estrutura lista.

ParÃ¢metros

n	numero que é colocado na lista

Retorna

Lista* Ponteiro para o novo elemento da lista.

```
00041                           {
00042
00043     Lista* aux = (Lista*)malloc(sizeof(Lista) * 1); // Aloca memória para um novo elemento na lista.
00044
00045     if (aux != NULL)
00046     {
00047         aux->num = n;      // Define o valor do número.
00048         aux->prox = NULL; // Define o proximo elemento como NULL.
00049     }
00050
00051     return aux; // Retorna uma apontador para o novo elemento da lista.
00052     free(aux);
00053 }
```

5.2.2.6 DistribuirDados()

```
Matriz * DistribuirDados (
            int linhas,
            int colunas )
```

Distribui os dados lidos de um arquivo em uma matriz dinâmica.

ParÃ¢metros

linhas	Número de linhas da matriz a ser criada.
colunas	Número de colunas por linha na matriz.

```
00144 {
00145     int i = 0;
00146     int quantosNum = 0;
00147     int* valores = CarregaDados(&quantosNum); // Carrega os dados do arquivo.
00148
00149     Matriz* matriz = NULL;
00150
00151     // Cria a matriz linha por linha.
00152     for (int l = 0; l < linhas; l++) {
00153         Lista* linha = NULL;
00154
00155         // Adiciona os valores à linha atual.
```

```
00156         for (int c = 0; c < colunas; c++) {
00157             linha = ColocarNaLista(linha, valores[i++]);
00158         }
00159 
00160         // Adiciona a linha à matriz.
00161         matriz = ColocarNaMatriz(matriz, linha);
00162     }
00163 
00164     return matriz; // Retorna a matriz criada.
00165 }
```

5.2.2.7 InserirColuna()

```
Matriz * InserirColuna (
            Matriz * matriz,
            int posicao,
            int * valores )
```

Insere uma nova coluna em uma posição específica em todas as linhas da matriz.
A função percorre cada linha da matriz, inserindo um novo valor da matriz de valores fornecida na posição especificada.

ParÂ¢metros

matriz	Apontador para a matriz onde a coluna será inserida.
posicao	Posição onde a nova coluna será inserida.
valores	Apontador para um arrays que prenche a nova coluna.

```
00284                                                                   {
00285 
00286     int contadorLinha = 0; // Contador para acessar o valor correspondente na matriz.
00287     int i = 0; // Indice para percorrer os valores a serem inseridos.
00288 
00289     Matriz* auxMatriz = matriz; // Apontador para percorrer a matriz.
00290 
00291 
00292     while (auxMatriz != NULL) // Percorre todas as linhas da matriz.
00293     {
00294         Lista* novaLinha = NULL; // Inicializa a nova configuração da linha.
00295         Lista* linhaAtual = auxMatriz->linha; // Apontador para percorrer os elementos da linha atual.
00296         int contadorColuna = 1; // Contador para encontrar a posição correta da coluna.
00297 
00298 
00299         while (linhaAtual != NULL && contadorColuna < posicao) // Copia os elementos da linha até a
      posição de inserção da nova coluna.
00300         {
00301             novaLinha = ColocarNaLista(novaLinha, linhaAtual->num);
00302             linhaAtual = linhaAtual->prox;
00303             contadorColuna++;
00304         }
00305 
00306 
00307         novaLinha = ColocarNaLista(novaLinha, valores[i++]); // Insere o novo valor na coluna
      especificada.
00308 
00309 
00310         while (linhaAtual != NULL) // Continua a copiar o restante dos elementos da linha.
00311         {
00312             novaLinha = ColocarNaLista(novaLinha, linhaAtual->num);
00313             linhaAtual = linhaAtual->prox;
00314         }
00315 
00316 
00317         auxMatriz->linha = novaLinha; // Atualiza a linha na matriz.
00318 
00319 
00320         auxMatriz = auxMatriz->ProxLinha; // Avança para a próxima linha da matriz.
00321         contadorLinha++;
00322     }
00323 
00324     return matriz; // Retorna o apontador para a matriz atualizada.
00325 }
```

5.2.2.8 InserirLinha()

```
Matriz * InserirLinha (
            Matriz * matriz,
```

```
            int linha,
            int tamanhoC,
            int * valores )
```

Insere uma nova linha após uma linha específica na matriz.
Esta função insere uma nova linha preenchida com os valores fornecidos após a linha especificada. Se a linha especificada for a última, a nova linha será adicionada ao final da matriz.

ParÃ¢metros

matriz	Aponta para a matriz onde a linha será inserida.
linha	Posição após a qual a nova linha será inserida.
tamanhoC	Numero de colunas na nova linha.
valores	Apontador para um arrays que prenche a nova linha.

```
00244                                                                                {
00245
00246     Matriz* LinhaM = matriz; // Inicializa o apontador para percorrer a matriz.
00247     int contadorLinha = 2; // Contador para encontrar a posição correta da linha.
00248
00249     // Percorre a matriz até encontrar a posição para inserir a nova linha.
00250     while (LinhaM != NULL && contadorLinha < linha)
00251     {
00252         LinhaM = LinhaM->ProxLinha; // Avança para a próxima linha.
00253         contadorLinha++;            // Incrementa o contador de linhas.
00254     }
00255     // Verifica se a posição foi encontrada.
00256     if (LinhaM != NULL)
00257     {
00258         Lista* novaLinha = NULL; // Inicializa a nova linha a ser inserida.
00259
00260         // Preenche a nova linha com os valores fornecidos.
00261         for (int i = 0; i < tamanhoC; i++)
00262         {
00263             novaLinha = ColocarNaLista(novaLinha, valores[i]);
00264         }
00265
00266
00267         Matriz* novaMatriz = CriarMatriz(novaLinha); // Cria um novo elemento de matriz para a nova
    linha.
00268         novaMatriz->ProxLinha = LinhaM->ProxLinha; // Insere a nova linha na posição correta.
00269         LinhaM->ProxLinha = novaMatriz; // Conecta a nova linha à matriz.
00270     }
00271
00272     return matriz; // Retorna o apontador para a matriz atualizada.
00273 }
```

5.2.2.9 MostrarMatriz()

```
void MostrarMatriz (
            Matriz * m )
```

Mostra os elementos de uma matriz no console.

ParÃ¢metros

m	Apontador para a primeira linha da matriz.

```
00173 {
00174     Matriz* linhaAtual = m;
00175
00176     // Corre cada linha da matriz.
00177     while (linhaAtual != NULL)
00178     {
00179         Lista* elementoAtual = linhaAtual->linha;
00180
00181         // Corre cada coluna da linha atual.
00182         while (elementoAtual != NULL)
00183         {
00184             printf("%d ", elementoAtual->num); // Imprime o valor do elemento.
00185             elementoAtual = elementoAtual->prox; // Passa para a proxima linha
00186         }
00187
00188         printf("\n"); // Nova linha após terminar de imprimir uma linha da matriz.
00189         linhaAtual = linhaAtual->ProxLinha;
00190     }
```

```
00191 }
```

5.2.2.10 MudarValor()

```
bool MudarValor (
            Matriz * matriz,
            int linha,
            int coluna,
            int ValorMudar )
```

Modifica o valor de um elemento específico na matriz.

ParÃ¢metros

matriz	Apontador para a matriz a ser modificada.
linha	Número da linha do elemento a ser modificado.
coluna	Número da coluna do elemento a ser modificado.
ValorMudar	Novo valor para o elemento especificado.

```
00202 {
00203     Matriz* auxLinha = matriz;
00204     int contadorLinha = 1;
00205
00206     // Encontra a linha especificada.
00207     while (auxLinha != NULL && contadorLinha < linha) {
00208         auxLinha = auxLinha->ProxLinha;
00209         contadorLinha++;
00210     }
00211
00212     if (auxLinha != NULL)
00213     {
00214         Lista* auxColuna = auxLinha->linha;
00215         int contadorColuna = 1;
00216
00217         // Encontra a coluna especificada.
00218         while (auxColuna != NULL && contadorColuna < coluna)
00219         {
00220             auxColuna = auxColuna->prox;
00221             contadorColuna++;
00222         }
00223
00224         if (auxColuna != NULL)
00225         {
00226             auxColuna->num = ValorMudar; // Modifica o valor.
00227             return true;                 // Retorna verdadeiro se mudar o valor.
00228         }
00229     }
00230     return false; // Retorna falso se não modificar o valor.
00231 }
```

5.2.2.11 RemoverColuna()

```
Matriz * RemoverColuna (
            Matriz * matriz,
            int coluna )
```

Remove uma coluna específica de todas as linhas da matriz.
Percorre todas as linhas da matriz, remove o elemento que corresponde à coluna especificada em cada linha.

ParÃ¢metros

matriz	Apontador para a matriz da qual a coluna será removida.
coluna	Número da coluna a ser removida

```
00379                                                {
00380
00381     Matriz* LinhaAtual = matriz; // Apontador para percorrer a matriz.
00382
00383     // Percorre todas as linhas da matriz.
00384     while (LinhaAtual != NULL)
00385     {
00386         Lista* ElementoAtual = LinhaAtual->linha; // Apontador para percorrer os elementos da linha.
```

```
00387         Lista* ElementoAnterior = NULL; // Mantém a referência ao elemento anterior na lista.
00388         int contadorColuna = 1; // Contador para encontrar a coluna especificada.
00389 
00390         // Percorre os elementos da linha até encontrar a coluna a ser removida.
00391         while (ElementoAtual != NULL)
00392         {
00393             if (contadorColuna == coluna)  // Verifica a coluna a ser removida.
00394             {
00395                 if (ElementoAnterior == NULL)  // Se for o primeiro elemento da linha.
00396                 {
00397                     LinhaAtual->linha = ElementoAtual->prox; // Atualiza o início da linha.
00398                 }
00399                 else
00400                 {
00401                     ElementoAnterior->prox = ElementoAtual->prox; // Remove o elemento da linha.
00402                 }
00403                 free(ElementoAtual); // Limpa a memória usada pelo elemento removido.
00404                 break;               // Sai do loop após remover o elemento.
00405             }
00406 
00407             ElementoAnterior = ElementoAtual;    // Atualiza o elemento anterior.
00408             ElementoAtual = ElementoAtual->prox; // Avança para o próximo elemento.
00409             contadorColuna++;                    // Incrementa o contador de colunas.
00410         }
00411 
00412         LinhaAtual = LinhaAtual->ProxLinha; // Avança para a proxima linha da matriz.
00413     }
00414 
00415     return matriz; // Retorna o ponteiro para a matriz atualizada.
00416 }
```

5.2.2.12 RemoverLinha()

```
Matriz * RemoverLinha (
            Matriz * matriz,
            int linha )
```

Remove uma linha específica da matriz.
Localiza a linha a ser removida. Se encontrada, a linha á removida.

ParÃ¢metros

matriz	Apontador para a matriz da qual a linha será removida.
linha	Número da linha a ser removida.

```
00336                                                {
00337 
00338     Matriz* LinhaAtual = matriz; // Apontador para percorrer a matriz.
00339     Matriz* LinhaAnterior = NULL; // Apontador para manter a linha anterior.
00340     int contadorLinha = 1; // Contador para encontrar a linha especificada.
00341 
00342     // Percorre a matriz até encontrar a linha a ser removida.
00343     while (LinhaAtual != NULL && contadorLinha < linha)
00344     {
00345         LinhaAnterior = LinhaAtual;         // Atualiza a linha anterior.
00346         LinhaAtual = LinhaAtual->ProxLinha; // Avança para a próxima linha.
00347         contadorLinha++;                    // Incrementa o contador de linhas.
00348     }
00349 
00350     if (LinhaAtual != NULL)  // Verifica se a linha a ser removida foi encontrada.
00351     {
00352         if (LinhaAnterior == NULL)  // Se for a primeira linha da matriz.
00353         {
00354             matriz = LinhaAtual->ProxLinha; // Atualiza o início da matriz.
00355         }
00356         else
00357         {
00358             LinhaAnterior->ProxLinha = LinhaAtual->ProxLinha; // Remove a linha da matriz.
00359         }
00360 
00361 
00362         Lista* elementoAtual = LinhaAtual->linha;
00363         Lista* tempElemento;
00364 
00365     }
00366 
00367     return matriz; // Retorna o apontador para a matriz atualizada.
00368 }
```

5.2.2.13 somaMaiores()

```
int somaMaiores (
            Matriz * m )
```

Soma os valores maiores da linha.
Percorre todas as linhas da matriz, e identifica o maior de todas as linhas.

ParÃ¢metros

matriz	Apontador para a matriz de qual iremos somar as linhas.

```
00424                           {
00425
00426
00427     Matriz* Linhas = m; // Um apontador para percorrer as linhas da matriz.
00428     int total = 0; // Acumular a soma dos maiores valores encontrados.
00429     int contadorColunas = 0; // Declara uma variável para contar as colunas
00430     int maior = 0;
00431
00432     // Enquanto houver linhas na matriz são percorridas.
00433     while (Linhas != NULL)
00434     {
00435         // Inicializa um apontador 'colunas' para percorrer os elementos (valores) de cada linha.
00436         Lista* colunas = Linhas->linha;
00437         contadorColunas = 0; // Reinicia o contador de colunas
00438         maior = 0; // Inicializa a 0 o maior no inicio de cada linha
00439
00440         // Enquanto houver elementos percorre
00441         while (colunas != NULL)
00442         {
00443             // Condição que derfine o maior da linha
00444             if (colunas->num > maior)
00445             {
00446                 maior = colunas->num; // Determina o maior das linhas
00447                 contadorColunas++; // Conta as colunas
00448             }
00449             colunas = colunas->prox; // Avança de coluna
00450         }
00451
00452         total += maior;// Adiciona o maior valor encontrado na linha
00453
00454         Linhas = Linhas->ProxLinha; // Avança para a proxima linha
00455     }
00456
00457
00458     return total; // Retorna o total acumulado dos maiores valores encontrados em cada linha.
00459 }
```

5.3 header.h

Ir para a documentação deste ficheiro.

```
00001 #pragma once
00002 #include <stdbool.h>
00003
00004 /*
00005 @struct Lista
00006 * @brief Esta estrutura para criar uma lista
00007 */
00008 typedef struct Lista
00009 {
00010     int num;
00011     struct Lista* prox;
00013 }Lista;
00014
00015 /*
00016 @struct Matriz
00017 * @brief Esta estrutura para criar uma Matriz
00018 */
00019 typedef struct Matriz
00020 {
00021     Lista* linha;
00022     struct Matriz* ProxLinha;
00024 }Matriz;
00025
00032 int* CarregaDados(int* tamanho);
00039 Lista* CriarNumero(int n);
00046 Lista* ColocarNaLista(Lista* inicio, int n);
00052 Matriz* CriarMatriz(Lista* lista);
00059 Matriz* ColocarNaMatriz(Matriz* inicio, Lista* linha);
```

```
00066 Matriz* DistribuirDados(int linhas, int colunas);
00072 void MostrarMatriz(Matriz* m);
00081 bool MudarValor(Matriz* matriz, int linha, int coluna, int ValorMudar);
00094 Matriz* InserirLinha(Matriz* matriz, int linha, int tamanhoC, int* valores);
00105 Matriz* InserirColuna(Matriz* matriz, int posicao, int* valores);
00115 Matriz* RemoverLinha(Matriz* matriz, int linha);
00125 Matriz* RemoverColuna(Matriz* matriz, int coluna);
00133 int somaMaiores(Matriz* m);
00134
```

5.4 ReferÃªncia ao ficheiro C:/Users/hugoc/OneDrive - Instituto PolitÃ©cnico do CÃ¡vado e do Ave/2023_2024/Estruturas de Dados AvanÃ§adas/Dev/Listas ligadas - Fase 1/src/ResoluÃ§Ã£o/Lib/header.h

```
#include <stdbool.h>
```

Estruturas de Dados

- struct Lista
- struct Matriz

DefiniÃ§Ãµes de tipos

- typedef struct Lista Lista
- typedef struct Matriz Matriz

FunÃ§Ãµes

- int * CarregaDados (int *tamanho)

  Carrega dados de um ficheiro CSV.
- Lista * CriarNumero (int n)

  Cria memória para alocar a estrutura lista.
- Lista * ColocarNaLista (Lista *inicio, int n)

  Insere um número no final da lista.
- Matriz * CriarMatriz (Lista *lista)

  Cria uma Matriz.
- Matriz * ColocarNaMatriz (Matriz *inicio, Lista *linha)

  Insere uma nova linha na matriz.
- Matriz * DistribuirDados (int linhas, int colunas)

  Distribui os dados lidos de um arquivo em uma matriz dinâmica.
- void MostrarMatriz (Matriz *m)

  Mostra os elementos de uma matriz no console.
- bool MudarValor (Matriz *matriz, int linha, int coluna, int ValorMudar)

  Modifica o valor de um elemento específico na matriz.
- Matriz * InserirLinha (Matriz *matriz, int linha, int tamanhoC, int *valores)

  Insere uma nova linha após uma linha específica na matriz.
- Matriz * InserirColuna (Matriz *matriz, int posicao, int *valores)

  Insere uma nova coluna em uma posição específica em todas as linhas da matriz.
- Matriz * RemoverLinha (Matriz *matriz, int linha)

  Remove uma linha específica da matriz.
- Matriz * RemoverColuna (Matriz *matriz, int coluna)

  Remove uma coluna específica de todas as linhas da matriz.
- int somaMaiores (Matriz *m)

  Soma os valores maiores da linha.

5.4.1 DocumentaÃ§Ã£o dos tipos

5.4.1.1 Lista

```
typedef struct Lista Lista
```

5.4.1.2 Matriz

```
typedef struct Matriz Matriz
```

5.4.2 DocumentaÃ§Ã£o das funÃ§Ãµes

5.4.2.1 CarregaDados()

```
int * CarregaDados (
            int * tamanho )
```

Carrega dados de um ficheiro CSV.

ParÃ¢metros

tamanho	Guarda quantos dados foram lidos

Retorna

int* Apontador que retorna os dados lidos.

```
00014                                {
00015     int i = 0;
00016     int* aux = (int*)malloc(sizeof(int) * 50); // Aloca memória para 50 inteiros.
00017
00018     FILE* ficheiro = fopen("matriz.csv", "r"); // Abrir o arquivo.
00019
00020     if (ficheiro == NULL) return NULL; // Retorna NULL caso o ficheiro não seja aberto.
00021
00022     // Lê os dados separados por ponto e virgula enquanto houver dados.
00023     while (fscanf_s(ficheiro, "%d;", &aux[i]) == 1)
00024     {
00025         i++;
00026     }
00027
00028     *tamanho = i;      // Armazena a quantidade de numeros lidos.
00029     fclose(ficheiro); // Fecha o arquivo.
00030
00031     return aux; // Retorna um apontador para os inteiros armazenados.
00032     free(aux);
00033 }
```

5.4.2.2 ColocarNaLista()

```
Lista * ColocarNaLista (
            Lista * inicio,
            int n )
```

Insere um número no final da lista.

ParÃ¢metros

inicio	Apontador para o primeiro elemento da lista.
n	O número a ser inserido na lista.

```
00062 {
00063     Lista* nova = CriarNumero(n); // Cria um novo elemento.
00064     if (nova == NULL)
00065         return inicio; // Retorna o início.
00066
00067     if (inicio == NULL)
00068     {
00069         inicio = nova; // Se a lista está vazia, o novo elemento é colocado no início.
00070     }
00071
00072     else
```

```
00073     {
00074         Lista* aux2 = inicio;
00075         // Percorre a lista até o final.
00076         while (aux2->prox != NULL)
00077         {
00078             aux2 = aux2->prox;
00079         }
00080         aux2->prox = nova; // Insere o novo elemento no final da lista.
00081     }
00082
00083     return inicio; // Retorna um apontador para o primeiro elemento da lista.
00084 }
```

5.4.2.3 ColocarNaMatriz()

```
Matriz * ColocarNaMatriz (
            Matriz * inicio,
            Lista * linha )
```

Insere uma nova linha na matriz.

ParÃ¢metros

inicio	Apontador para a primeira linha da matriz.
linha	Apontador para a lista que será inserida como nova linha.

```
00112 {
00113
00114     Matriz* nova = CriarMatriz(linha); // Cria uma nova linha com a lista fornecida.
00115
00116     if (nova == NULL) return inicio; // Retorna o início se falha ao criar a nova linha.
00117
00118     if (inicio == NULL)
00119     {
00120         inicio = nova; // Se a matriz está vazia, a nova linha é o início.
00121     }
00122
00123     else
00124     {
00125         Matriz* aux2 = inicio;
00126
00127         // Percorre a matriz até a última linha.
00128         while (aux2->ProxLinha != NULL)
00129         {
00130             aux2 = aux2->ProxLinha;
00131         }
00132         aux2->ProxLinha = nova; // Insere a nova linha no final da matriz.
00133     }
00134     return inicio; // Retorna um apontador para a primeira linha da matriz.
00135 }
```

5.4.2.4 CriarMatriz()

```
Matriz * CriarMatriz (
            Lista * lista )
```

Cria uma Matriz.

ParÃ¢metros

lista	Apontador para a lista que é uma linha da matriz.

```
00091                                   {
00092
00093     Matriz* aux = (Matriz*)malloc(sizeof(Matriz) * 1); // Aloca memória para uma nova linha da matriz.
00094
00095     if (aux != NULL)
00096     {
00097         aux->linha = lista;     // Define a lista como linha da matriz.
00098         aux->ProxLinha = NULL; // Define a próxima linha vazia
00099     }
00100
00101     return aux; // Retorna um apontador para a nova linha da matriz.
00102     free(aux);
00103 }
```

5.4.2.5 CriarNumero()

```
Lista * CriarNumero (
            int n )
```

Cria memória para alocar a estrutura lista.

ParÃ¢metros

n	numero que é colocado na lista

Retorna

Lista* Ponteiro para o novo elemento da lista.

```
00041                           {
00042
00043     Lista* aux = (Lista*)malloc(sizeof(Lista) * 1); // Aloca memória para um novo elemento na lista.
00044
00045     if (aux != NULL)
00046     {
00047         aux->num = n;      // Define o valor do número.
00048         aux->prox = NULL; // Define o proximo elemento como NULL.
00049     }
00050
00051     return aux; // Retorna uma apontador para o novo elemento da lista.
00052     free(aux);
00053 }
```

5.4.2.6 DistribuirDados()

```
Matriz * DistribuirDados (
            int linhas,
            int colunas )
```

Distribui os dados lidos de um arquivo em uma matriz dinâmica.

ParÃ¢metros

linhas	Número de linhas da matriz a ser criada.
colunas	Número de colunas por linha na matriz.

```
00144 {
00145     int i = 0;
00146     int quantosNum = 0;
00147     int* valores = CarregaDados(&quantosNum); // Carrega os dados do arquivo.
00148
00149     Matriz* matriz = NULL;
00150
00151     // Cria a matriz linha por linha.
00152     for (int l = 0; l < linhas; l++) {
00153         Lista* linha = NULL;
00154
00155         // Adiciona os valores à linha atual.
00156         for (int c = 0; c < colunas; c++) {
00157             linha = ColocarNaLista(linha, valores[i++]);
00158         }
00159
00160         // Adiciona a linha à matriz.
00161         matriz = ColocarNaMatriz(matriz, linha);
00162     }
00163
00164     return matriz; // Retorna a matriz criada.
00165 }
```

5.4.2.7 InserirColuna()

```
Matriz * InserirColuna (
            Matriz * matriz,
            int posicao,
            int * valores )
```

Insere uma nova coluna em uma posição específica em todas as linhas da matriz.

A função percorre cada linha da matriz, inserindo um novo valor da matriz de valores fornecida na posição especificada.

ParÃ¢metros

matriz	Apontador para a matriz onde a coluna será inserida.
posicao	Posição onde a nova coluna será inserida.
valores	Apontador para um arrays que prenche a nova coluna.

```
00284                                                                {
00285
00286     int contadorLinha = 0; // Contador para acessar o valor correspondente na matriz.
00287     int i = 0; // Indice para percorrer os valores a serem inseridos.
00288
00289     Matriz* auxMatriz = matriz; // Apontador para percorrer a matriz.
00290
00291
00292     while (auxMatriz != NULL) // Percorre todas as linhas da matriz.
00293     {
00294         Lista* novaLinha = NULL; // Inicializa a nova configuração da linha.
00295         Lista* linhaAtual = auxMatriz->linha; // Apontador para percorrer os elementos da linha atual.
00296         int contadorColuna = 1; // Contador para encontrar a posição correta da coluna.
00297
00298
00299         while (linhaAtual != NULL && contadorColuna < posicao) // Copia os elementos da linha até a
      posição de inserção da nova coluna.
00300         {
00301             novaLinha = ColocarNaLista(novaLinha, linhaAtual->num);
00302             linhaAtual = linhaAtual->prox;
00303             contadorColuna++;
00304         }
00305
00306
00307         novaLinha = ColocarNaLista(novaLinha, valores[i++]); // Insere o novo valor na coluna
      especificada.
00308
00309
00310         while (linhaAtual != NULL) // Continua a copiar o restante dos elementos da linha.
00311         {
00312             novaLinha = ColocarNaLista(novaLinha, linhaAtual->num);
00313             linhaAtual = linhaAtual->prox;
00314         }
00315
00316
00317         auxMatriz->linha = novaLinha; // Atualiza a linha na matriz.
00318
00319
00320         auxMatriz = auxMatriz->ProxLinha; // Avança para a próxima linha da matriz.
00321         contadorLinha++;
00322     }
00323
00324     return matriz; // Retorna o apontador para a matriz atualizada.
00325 }
```

5.4.2.8 InserirLinha()

```
Matriz * InserirLinha (
            Matriz * matriz,
            int linha,
            int tamanhoC,
            int * valores )
```

Insere uma nova linha após uma linha específica na matriz.
Esta função insere uma nova linha preenchida com os valores fornecidos após a linha especificada. Se a linha especificada for a última, a nova linha será adicionada ao final da matriz.

ParÃ¢metros

matriz	Aponta para a matriz onde a linha será inserida.
linha	Posição após a qual a nova linha será inserida.
tamanhoC	Numero de colunas na nova linha.
valores	Apontador para um arrays que prenche a nova linha.

```
00244                                                                                {
00245
00246     Matriz* LinhaM = matriz; // Inicializa o apontador para percorrer a matriz.
00247     int contadorLinha = 2; // Contador para encontrar a posição correta da linha.
00248
00249     // Percorre a matriz até encontrar a posição para inserir a nova linha.
00250     while (LinhaM != NULL && contadorLinha < linha)
00251     {
00252         LinhaM = LinhaM->ProxLinha; // Avança para a próxima linha.
00253         contadorLinha++;            // Incrementa o contador de linhas.
00254     }
00255     // Verifica se a posição foi encontrada.
00256     if (LinhaM != NULL)
00257     {
00258         Lista* novaLinha = NULL; // Inicializa a nova linha a ser inserida.
00259
00260         // Preenche a nova linha com os valores fornecidos.
00261         for (int i = 0; i < tamanhoC; i++)
00262         {
00263             novaLinha = ColocarNaLista(novaLinha, valores[i]);
00264         }
00265
00266
00267         Matriz* novaMatriz = CriarMatriz(novaLinha); // Cria um novo elemento de matriz para a nova
      linha.
00268         novaMatriz->ProxLinha = LinhaM->ProxLinha; // Insere a nova linha na posição correta.
00269         LinhaM->ProxLinha = novaMatriz; // Conecta a nova linha à matriz.
00270     }
00271
00272     return matriz; // Retorna o apontador para a matriz atualizada.
00273 }
```

5.4.2.9 MostrarMatriz()

```
void MostrarMatriz (
            Matriz * m )
```

Mostra os elementos de uma matriz no console.

ParÃ¢metros

m	Apontador para a primeira linha da matriz.

```
00173 {
00174     Matriz* linhaAtual = m;
00175
00176     // Corre cada linha da matriz.
00177     while (linhaAtual != NULL)
00178     {
00179         Lista* elementoAtual = linhaAtual->linha;
00180
00181         // Corre cada coluna da linha atual.
00182         while (elementoAtual != NULL)
00183         {
00184             printf("%d ", elementoAtual->num); // Imprime o valor do elemento.
00185             elementoAtual = elementoAtual->prox; // Passa para a proxima linha
00186         }
00187
00188         printf("\n"); // Nova linha após terminar de imprimir uma linha da matriz.
00189         linhaAtual = linhaAtual->ProxLinha;
00190     }
00191 }
```

5.4.2.10 MudarValor()

```
bool MudarValor (
            Matriz * matriz,
            int linha,
            int coluna,
            int ValorMudar )
```

Modifica o valor de um elemento específico na matriz.

ParÃ¢metros

matriz	Apontador para a matriz a ser modificada.
linha	Número da linha do elemento a ser modificado.

ParÃ¢metros

coluna	Número da coluna do elemento a ser modificado.
ValorMudar	Novo valor para o elemento especificado.

```
00202 {
00203     Matriz* auxLinha = matriz;
00204     int contadorLinha = 1;
00205 
00206     // Encontra a linha especificada.
00207     while (auxLinha != NULL && contadorLinha < linha) {
00208         auxLinha = auxLinha->ProxLinha;
00209         contadorLinha++;
00210     }
00211 
00212     if (auxLinha != NULL)
00213     {
00214         Lista* auxColuna = auxLinha->linha;
00215         int contadorColuna = 1;
00216 
00217         // Encontra a coluna especificada.
00218         while (auxColuna != NULL && contadorColuna < coluna)
00219         {
00220             auxColuna = auxColuna->prox;
00221             contadorColuna++;
00222         }
00223 
00224         if (auxColuna != NULL)
00225         {
00226             auxColuna->num = ValorMudar; // Modifica o valor.
00227             return true;                 // Retorna verdadeiro se mudar o valor.
00228         }
00229     }
00230     return false; // Retorna falso se não modificar o valor.
00231 }
```

5.4.2.11 RemoverColuna()

```
Matriz * RemoverColuna (
            Matriz * matriz,
            int coluna )
```

Remove uma coluna específica de todas as linhas da matriz.
Percorre todas as linhas da matriz, remove o elemento que corresponde à coluna especificada em cada linha.

ParÃ¢metros

matriz	Apontador para a matriz da qual a coluna será removida.
coluna	Número da coluna a ser removida

```
00379                                                {
00380 
00381     Matriz* LinhaAtual = matriz; // Apontador para percorrer a matriz.
00382 
00383     // Percorre todas as linhas da matriz.
00384     while (LinhaAtual != NULL)
00385     {
00386         Lista* ElementoAtual = LinhaAtual->linha; // Apontador para percorrer os elementos da linha.
00387         Lista* ElementoAnterior = NULL; // Mantém a referência ao elemento anterior na lista.
00388         int contadorColuna = 1; // Contador para encontrar a coluna especificada.
00389 
00390         // Percorre os elementos da linha até encontrar a coluna a ser removida.
00391         while (ElementoAtual != NULL)
00392         {
00393             if (contadorColuna == coluna)  // Verifica a coluna a ser removida.
00394             {
00395                 if (ElementoAnterior == NULL)  // Se for o primeiro elemento da linha.
00396                 {
00397                     LinhaAtual->linha = ElementoAtual->prox; // Atualiza o início da linha.
00398                 }
00399                 else
00400                 {
00401                     ElementoAnterior->prox = ElementoAtual->prox; // Remove o elemento da linha.
00402                 }
00403                 free(ElementoAtual); // Limpa a memória usada pelo elemento removido.
00404                 break;               // Sai do loop após remover o elemento.
00405             }
00406 
```

```
00407             ElementoAnterior = ElementoAtual;    // Atualiza o elemento anterior.
00408             ElementoAtual = ElementoAtual->prox; // Avança para o próximo elemento.
00409             contadorColuna++;                    // Incrementa o contador de colunas.
00410         }
00411
00412         LinhaAtual = LinhaAtual->ProxLinha; // Avança para a proxima linha da matriz.
00413     }
00414
00415     return matriz; // Retorna o ponteiro para a matriz atualizada.
00416 }
```

5.4.2.12 RemoverLinha()

```
Matriz * RemoverLinha (
            Matriz * matriz,
            int linha )
```

Remove uma linha específica da matriz.
Localiza a linha a ser removida. Se encontrada, a linha á removida.

ParÃ¢metros

matriz	Apontador para a matriz da qual a linha será removida.
linha	Número da linha a ser removida.

```
00336                                                {
00337
00338     Matriz* LinhaAtual = matriz; // Apontador para percorrer a matriz.
00339     Matriz* LinhaAnterior = NULL; // Apontador para manter a linha anterior.
00340     int contadorLinha = 1; // Contador para encontrar a linha especificada.
00341
00342     // Percorre a matriz até encontrar a linha a ser removida.
00343     while (LinhaAtual != NULL && contadorLinha < linha)
00344     {
00345         LinhaAnterior = LinhaAtual;         // Atualiza a linha anterior.
00346         LinhaAtual = LinhaAtual->ProxLinha; // Avança para a próxima linha.
00347         contadorLinha++;                    // Incrementa o contador de linhas.
00348     }
00349
00350     if (LinhaAtual != NULL)  // Verifica se a linha a ser removida foi encontrada.
00351     {
00352         if (LinhaAnterior == NULL)  // Se for a primeira linha da matriz.
00353         {
00354             matriz = LinhaAtual->ProxLinha; // Atualiza o início da matriz.
00355         }
00356         else
00357         {
00358             LinhaAnterior->ProxLinha = LinhaAtual->ProxLinha; // Remove a linha da matriz.
00359         }
00360
00361
00362         Lista* elementoAtual = LinhaAtual->linha;
00363         Lista* tempElemento;
00364
00365     }
00366
00367     return matriz; // Retorna o apontador para a matriz atualizada.
00368 }
```

5.4.2.13 somaMaiores()

```
int somaMaiores (
            Matriz * m )
```

Soma os valores maiores da linha.
Percorre todas as linhas da matriz, e identifica o maior de todas as linhas.

ParÃ¢metros

matriz	Apontador para a matriz de qual iremos somar as linhas.

```
00424                           {
00425
00426
00427     Matriz* Linhas = m; // Um apontador para percorrer as linhas da matriz.
```

```
00428     int total = 0; // Acumular a soma dos maiores valores encontrados.
00429     int contadorColunas = 0; // Declara uma variável para contar as colunas
00430     int maior = 0;
00431
00432     // Enquanto houver linhas na matriz são percorridas.
00433     while (Linhas != NULL)
00434     {
00435         // Inicializa um apontador 'colunas' para percorrer os elementos (valores) de cada linha.
00436         Lista* colunas = Linhas->linha;
00437         contadorColunas = 0; // Reinicia o contador de colunas
00438         maior = 0; // Inicializa a 0 o maior no inicio de cada linha
00439
00440         // Enquanto houver elementos percorre
00441         while (colunas != NULL)
00442         {
00443             // Condição que derfine o maior da linha
00444             if (colunas->num > maior)
00445             {
00446                 maior = colunas->num; // Determina o maior das linhas
00447                 contadorColunas++; // Conta as colunas
00448             }
00449             colunas = colunas->prox; // Avança de coluna
00450         }
00451
00452         total += maior;// Adiciona o maior valor encontrado na linha
00453
00454         Linhas = Linhas->ProxLinha; // Avança para a proxima linha
00455     }
00456
00457
00458     return total; // Retorna o total acumulado dos maiores valores encontrados em cada linha.
00459 }
```

5.5 header.h

Ir para a documentação deste ficheiro.

```
00001 #pragma once
00002 #include <stdbool.h>
00003
00004 /*
00005 @struct Lista
00006 * @brief Esta estrutura para criar uma lista
00007 */
00008 typedef struct Lista
00009 {
00010     int num;
00011     struct Lista* prox;
00013 }Lista;
00014
00015 /*
00016 @struct Matriz
00017 * @brief Esta estrutura para criar uma Matriz
00018 */
00019 typedef struct Matriz
00020 {
00021     Lista* linha;
00022     struct Matriz* ProxLinha;
00024 }Matriz;
00025
00032 int* CarregaDados(int* tamanho);
00039 Lista* CriarNumero(int n);
00046 Lista* ColocarNaLista(Lista* inicio, int n);
00052 Matriz* CriarMatriz(Lista* lista);
00059 Matriz* ColocarNaMatriz(Matriz* inicio, Lista* linha);
00066 Matriz* DistribuirDados(int linhas, int colunas);
00072 void MostrarMatriz(Matriz* m);
00081 bool MudarValor(Matriz* matriz, int linha, int coluna, int ValorMudar);
00094 Matriz* InserirLinha(Matriz* matriz, int linha, int tamanhoC, int* valores);
00105 Matriz* InserirColuna(Matriz* matriz, int posicao, int* valores);
00115 Matriz* RemoverLinha(Matriz* matriz, int linha);
00125 Matriz* RemoverColuna(Matriz* matriz, int coluna);
00133 int somaMaiores(Matriz* m);
00134
```

5.6 ReferÃªncia ao ficheiro C:/Users/hugoc/OneDrive - Instituto PolitÃ©cnico do CÃ¡vado e do Ave/2023_2024/Estruturas de Dados AvanÃ§adas/Dev/Listas ligadas - Fase 1/src/FunÃ§Ãµes/source.c

```
#include "header.h"
#include <locale.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
```

FunÃ§Ãµes

- int * CarregaDados (int *tamanho)

  Carrega dados de um ficheiro CSV.
- Lista * CriarNumero (int n)

  Cria memória para alocar a estrutura lista.
- Lista * ColocarNaLista (Lista *inicio, int n)

  Insere um número no final da lista.
- Matriz * CriarMatriz (Lista *lista)

  Cria uma Matriz.
- Matriz * ColocarNaMatriz (Matriz *inicio, Lista *linha)

  Insere uma nova linha na matriz.
- Matriz * DistribuirDados (int linhas, int colunas)

  Distribui os dados lidos de um arquivo em uma matriz dinâmica.
- void MostrarMatriz (Matriz *m)

  Mostra os elementos de uma matriz no console.
- bool MudarValor (Matriz *matriz, int linha, int coluna, int ValorMudar)

  Modifica o valor de um elemento específico na matriz.
- Matriz * InserirLinha (Matriz *matriz, int linha, int tamanhoC, int *valores)

  Insere uma nova linha após uma linha específica na matriz.
- Matriz * InserirColuna (Matriz *matriz, int posicao, int *valores)

  Insere uma nova coluna em uma posição específica em todas as linhas da matriz.
- Matriz * RemoverLinha (Matriz *matriz, int linha)

  Remove uma linha específica da matriz.
- Matriz * RemoverColuna (Matriz *matriz, int coluna)

  Remove uma coluna específica de todas as linhas da matriz.
- int somaMaiores (Matriz *m)

  Soma os valores maiores da linha.

5.6.1 DocumentaÃ§Ã£o das funÃ§Ãµes

5.6.1.1 CarregaDados()

```
int * CarregaDados (
            int * tamanho )
```

Carrega dados de um ficheiro CSV.

ParÃ¢metros

tamanho	Guarda quantos dados foram lidos

Retorna

int* Apontador que retorna os dados lidos.

```
00014                                   {
00015     int i = 0;
00016     int* aux = (int*)malloc(sizeof(int) * 50); // Aloca memória para 50 inteiros.
00017
00018     FILE* ficheiro = fopen("matriz.csv", "r"); // Abrir o arquivo.
00019
00020     if (ficheiro == NULL) return NULL; // Retorna NULL caso o ficheiro não seja aberto.
00021
00022     // Lê os dados separados por ponto e virgula enquanto houver dados.
00023     while (fscanf_s(ficheiro, "%d;", &aux[i]) == 1)
00024     {
00025         i++;
00026     }
00027
00028     *tamanho = i;     // Armazena a quantidade de numeros lidos.
00029     fclose(ficheiro); // Fecha o arquivo.
00030
00031     return aux; // Retorna um apontador para os inteiros armazenados.
00032     free(aux);
00033 }
```

5.6.1.2 ColocarNaLista()

```
Lista * ColocarNaLista (
            Lista * inicio,
            int n )
```

Insere um número no final da lista.

ParÃ¢metros

inicio	Apontador para o primeiro elemento da lista.
n	O número a ser inserido na lista.

```
00062 {
00063     Lista* nova = CriarNumero(n); // Cria um novo elemento.
00064     if (nova == NULL)
00065         return inicio; // Retorna o início.
00066
00067     if (inicio == NULL)
00068     {
00069         inicio = nova; // Se a lista está vazia, o novo elemento é colocado no início.
00070     }
00071
00072     else
00073     {
00074         Lista* aux2 = inicio;
00075         // Percorre a lista até o final.
00076         while (aux2->prox != NULL)
00077         {
00078             aux2 = aux2->prox;
00079         }
00080         aux2->prox = nova; // Insere o novo elemento no final da lista.
00081     }
00082
00083     return inicio; // Retorna um apontador para o primeiro elemento da lista.
00084 }
```

5.6.1.3 ColocarNaMatriz()

```
Matriz * ColocarNaMatriz (
            Matriz * inicio,
            Lista * linha )
```

Insere uma nova linha na matriz.

ParÃ¢metros

inicio	Apontador para a primeira linha da matriz.
linha	Apontador para a lista que será inserida como nova linha.

```
00112 {
00113
00114     Matriz* nova = CriarMatriz(linha); // Cria uma nova linha com a lista fornecida.
00115
00116     if (nova == NULL) return inicio; // Retorna o início se falha ao criar a nova linha.
00117
00118     if (inicio == NULL)
00119     {
00120         inicio = nova; // Se a matriz está vazia, a nova linha é o início.
00121     }
00122
00123     else
00124     {
00125         Matriz* aux2 = inicio;
00126
00127         // Percorre a matriz até a última linha.
00128         while (aux2->ProxLinha != NULL)
00129         {
00130             aux2 = aux2->ProxLinha;
00131         }
00132         aux2->ProxLinha = nova; // Insere a nova linha no final da matriz.
00133     }
00134     return inicio; // Retorna um apontador para a primeira linha da matriz.
00135 }
```

5.6.1.4 CriarMatriz()

```
Matriz * CriarMatriz (
            Lista * lista )
```

Cria uma Matriz.

ParÃ¢metros

lista	Apontador para a lista que é uma linha da matriz.

```
00091                                   {
00092
00093     Matriz* aux = (Matriz*)malloc(sizeof(Matriz) * 1); // Aloca memória para uma nova linha da matriz.
00094
00095     if (aux != NULL)
00096     {
00097         aux->linha = lista;    // Define a lista como linha da matriz.
00098         aux->ProxLinha = NULL; // Define a próxima linha vazia
00099     }
00100
00101     return aux; // Retorna um apontador para a nova linha da matriz.
00102     free(aux);
00103 }
```

5.6.1.5 CriarNumero()

```
Lista * CriarNumero (
            int n )
```

Cria memória para alocar a estrutura lista.

ParÃ¢metros

n	numero que é colocado na lista

Retorna

Lista* Ponteiro para o novo elemento da lista.

```
00041                           {
00042
00043     Lista* aux = (Lista*)malloc(sizeof(Lista) * 1); // Aloca memória para um novo elemento na lista.
00044
00045     if (aux != NULL)
00046     {
00047         aux->num = n;     // Define o valor do número.
00048         aux->prox = NULL; // Define o proximo elemento como NULL.
00049     }
00050
00051     return aux; // Retorna uma apontador para o novo elemento da lista.
```

```
00052     free(aux);
00053 }
```

5.6.1.6 DistribuirDados()

```
Matriz * DistribuirDados (
            int linhas,
            int colunas )
```

Distribui os dados lidos de um arquivo em uma matriz dinâmica.

ParÃ¢metros

linhas	Número de linhas da matriz a ser criada.
colunas	Número de colunas por linha na matriz.

```
00144 {
00145     int i = 0;
00146     int quantosNum = 0;
00147     int* valores = CarregaDados(&quantosNum); // Carrega os dados do arquivo.
00148
00149     Matriz* matriz = NULL;
00150
00151     // Cria a matriz linha por linha.
00152     for (int l = 0; l < linhas; l++) {
00153         Lista* linha = NULL;
00154
00155         // Adiciona os valores à linha atual.
00156         for (int c = 0; c < colunas; c++) {
00157             linha = ColocarNaLista(linha, valores[i++]);
00158         }
00159
00160         // Adiciona a linha à matriz.
00161         matriz = ColocarNaMatriz(matriz, linha);
00162     }
00163
00164     return matriz; // Retorna a matriz criada.
00165 }
```

5.6.1.7 InserirColuna()

```
Matriz * InserirColuna (
            Matriz * matriz,
            int posicao,
            int * valores )
```

Insere uma nova coluna em uma posição específica em todas as linhas da matriz.
A função percorre cada linha da matriz, inserindo um novo valor da matriz de valores fornecida na posição especificada.

ParÃ¢metros

matriz	Apontador para a matriz onde a coluna será inserida.
posicao	Posição onde a nova coluna será inserida.
valores	Apontador para um arrays que prenche a nova coluna.

```
00284                                                                   {
00285
00286     int contadorLinha = 0; // Contador para acessar o valor correspondente na matriz.
00287     int i = 0; // Indice para percorrer os valores a serem inseridos.
00288
00289     Matriz* auxMatriz = matriz; // Apontador para percorrer a matriz.
00290
00291
00292     while (auxMatriz != NULL) // Percorre todas as linhas da matriz.
00293     {
00294         Lista* novaLinha = NULL; // Inicializa a nova configuração da linha.
00295         Lista* linhaAtual = auxMatriz->linha; // Apontador para percorrer os elementos da linha atual.
00296         int contadorColuna = 1; // Contador para encontrar a posição correta da coluna.
00297
00298
00299         while (linhaAtual != NULL && contadorColuna < posicao) // Copia os elementos da linha até a
      posição de inserção da nova coluna.
```

```
00300         {
00301             novaLinha = ColocarNaLista(novaLinha, linhaAtual->num);
00302             linhaAtual = linhaAtual->prox;
00303             contadorColuna++;
00304         }
00305
00306
00307         novaLinha = ColocarNaLista(novaLinha, valores[i++]); // Insere o novo valor na coluna
      especificada.
00308
00309
00310         while (linhaAtual != NULL) // Continua a copiar o restante dos elementos da linha.
00311         {
00312             novaLinha = ColocarNaLista(novaLinha, linhaAtual->num);
00313             linhaAtual = linhaAtual->prox;
00314         }
00315
00316
00317         auxMatriz->linha = novaLinha; // Atualiza a linha na matriz.
00318
00319
00320         auxMatriz = auxMatriz->ProxLinha; // Avança para a próxima linha da matriz.
00321         contadorLinha++;
00322     }
00323
00324     return matriz; // Retorna o apontador para a matriz atualizada.
00325 }
```

5.6.1.8 InserirLinha()

```
Matriz * InserirLinha (
            Matriz * matriz,
            int linha,
            int tamanhoC,
            int * valores )
```

Insere uma nova linha após uma linha específica na matriz.
Esta função insere uma nova linha preenchida com os valores fornecidos após a linha especificada. Se a linha especificada for a última, a nova linha será adicionada ao final da matriz.

ParÃ¢metros

matriz	Aponta para a matriz onde a linha será inserida.
linha	Posição após a qual a nova linha será inserida.
tamanhoC	Numero de colunas na nova linha.
valores	Apontador para um arrays que prenche a nova linha.

```
00244                                                                                {
00245
00246     Matriz* LinhaM = matriz; // Inicializa o apontador para percorrer a matriz.
00247     int contadorLinha = 2; // Contador para encontrar a posição correta da linha.
00248
00249     // Percorre a matriz até encontrar a posição para inserir a nova linha.
00250     while (LinhaM != NULL && contadorLinha < linha)
00251     {
00252         LinhaM = LinhaM->ProxLinha; // Avança para a próxima linha.
00253         contadorLinha++;            // Incrementa o contador de linhas.
00254     }
00255     // Verifica se a posição foi encontrada.
00256     if (LinhaM != NULL)
00257     {
00258         Lista* novaLinha = NULL; // Inicializa a nova linha a ser inserida.
00259
00260         // Preenche a nova linha com os valores fornecidos.
00261         for (int i = 0; i < tamanhoC; i++)
00262         {
00263             novaLinha = ColocarNaLista(novaLinha, valores[i]);
00264         }
00265
00266
00267         Matriz* novaMatriz = CriarMatriz(novaLinha); // Cria um novo elemento de matriz para a nova
      linha.
00268         novaMatriz->ProxLinha = LinhaM->ProxLinha; // Insere a nova linha na posição correta.
00269         LinhaM->ProxLinha = novaMatriz; // Conecta a nova linha à matriz.
00270     }
00271
00272     return matriz; // Retorna o apontador para a matriz atualizada.
00273 }
```

5.6.1.9 MostrarMatriz()

```
void MostrarMatriz (
            Matriz * m )
```

Mostra os elementos de uma matriz no console.

ParÃ¢metros

m	Apontador para a primeira linha da matriz.

```
00173 {
00174     Matriz* linhaAtual = m;
00175
00176     // Corre cada linha da matriz.
00177     while (linhaAtual != NULL)
00178     {
00179         Lista* elementoAtual = linhaAtual->linha;
00180
00181         // Corre cada coluna da linha atual.
00182         while (elementoAtual != NULL)
00183         {
00184             printf("%d ", elementoAtual->num); // Imprime o valor do elemento.
00185             elementoAtual = elementoAtual->prox; // Passa para a proxima linha
00186         }
00187
00188         printf("\n"); // Nova linha após terminar de imprimir uma linha da matriz.
00189         linhaAtual = linhaAtual->ProxLinha;
00190     }
00191 }
```

5.6.1.10 MudarValor()

```
bool MudarValor (
            Matriz * matriz,
            int linha,
            int coluna,
            int ValorMudar )
```

Modifica o valor de um elemento específico na matriz.

ParÃ¢metros

matriz	Apontador para a matriz a ser modificada.
linha	Número da linha do elemento a ser modificado.
coluna	Número da coluna do elemento a ser modificado.
ValorMudar	Novo valor para o elemento especificado.

```
00202 {
00203     Matriz* auxLinha = matriz;
00204     int contadorLinha = 1;
00205
00206     // Encontra a linha especificada.
00207     while (auxLinha != NULL && contadorLinha < linha) {
00208         auxLinha = auxLinha->ProxLinha;
00209         contadorLinha++;
00210     }
00211
00212     if (auxLinha != NULL)
00213     {
00214         Lista* auxColuna = auxLinha->linha;
00215         int contadorColuna = 1;
00216
00217         // Encontra a coluna especificada.
00218         while (auxColuna != NULL && contadorColuna < coluna)
00219         {
00220             auxColuna = auxColuna->prox;
00221             contadorColuna++;
00222         }
00223
00224         if (auxColuna != NULL)
00225         {
00226             auxColuna->num = ValorMudar; // Modifica o valor.
00227             return true;                 // Retorna verdadeiro se mudar o valor.
00228         }
```

```
00229     }
00230     return false; // Retorna falso se não modificar o valor.
00231 }
```

5.6.1.11 RemoverColuna()

```
Matriz * RemoverColuna (
            Matriz * matriz,
            int coluna )
```

Remove uma coluna específica de todas as linhas da matriz.
Percorre todas as linhas da matriz, remove o elemento que corresponde à coluna especificada em cada linha.

ParÃ¢metros

matriz	Apontador para a matriz da qual a coluna será removida.
coluna	Número da coluna a ser removida

```
00379                                                {
00380
00381     Matriz* LinhaAtual = matriz; // Apontador para percorrer a matriz.
00382
00383     // Percorre todas as linhas da matriz.
00384     while (LinhaAtual != NULL)
00385     {
00386         Lista* ElementoAtual = LinhaAtual->linha; // Apontador para percorrer os elementos da linha.
00387         Lista* ElementoAnterior = NULL; // Mantém a referência ao elemento anterior na lista.
00388         int contadorColuna = 1; // Contador para encontrar a coluna especificada.
00389
00390         // Percorre os elementos da linha até encontrar a coluna a ser removida.
00391         while (ElementoAtual != NULL)
00392         {
00393             if (contadorColuna == coluna)  // Verifica a coluna a ser removida.
00394             {
00395                 if (ElementoAnterior == NULL)  // Se for o primeiro elemento da linha.
00396                 {
00397                     LinhaAtual->linha = ElementoAtual->prox; // Atualiza o início da linha.
00398                 }
00399                 else
00400                 {
00401                     ElementoAnterior->prox = ElementoAtual->prox; // Remove o elemento da linha.
00402                 }
00403                 free(ElementoAtual); // Limpa a memória usada pelo elemento removido.
00404                 break;               // Sai do loop após remover o elemento.
00405             }
00406
00407             ElementoAnterior = ElementoAtual;    // Atualiza o elemento anterior.
00408             ElementoAtual = ElementoAtual->prox; // Avança para o próximo elemento.
00409             contadorColuna++;                    // Incrementa o contador de colunas.
00410         }
00411
00412         LinhaAtual = LinhaAtual->ProxLinha; // Avança para a proxima linha da matriz.
00413     }
00414
00415     return matriz; // Retorna o ponteiro para a matriz atualizada.
00416 }
```

5.6.1.12 RemoverLinha()

```
Matriz * RemoverLinha (
            Matriz * matriz,
            int linha )
```

Remove uma linha específica da matriz.
Localiza a linha a ser removida. Se encontrada, a linha á removida.

ParÃ¢metros

matriz	Apontador para a matriz da qual a linha será removida.
linha	Número da linha a ser removida.

```
00336                                                {
00337
00338     Matriz* LinhaAtual = matriz; // Apontador para percorrer a matriz.
```

```
00339     Matriz* LinhaAnterior = NULL; // Apontador para manter a linha anterior.
00340     int contadorLinha = 1; // Contador para encontrar a linha especificada.
00341
00342     // Percorre a matriz até encontrar a linha a ser removida.
00343     while (LinhaAtual != NULL && contadorLinha < linha)
00344     {
00345         LinhaAnterior = LinhaAtual;         // Atualiza a linha anterior.
00346         LinhaAtual = LinhaAtual->ProxLinha; // Avança para a próxima linha.
00347         contadorLinha++;                    // Incrementa o contador de linhas.
00348     }
00349
00350     if (LinhaAtual != NULL)  // Verifica se a linha a ser removida foi encontrada.
00351     {
00352         if (LinhaAnterior == NULL)  // Se for a primeira linha da matriz.
00353         {
00354             matriz = LinhaAtual->ProxLinha; // Atualiza o início da matriz.
00355         }
00356         else
00357         {
00358             LinhaAnterior->ProxLinha = LinhaAtual->ProxLinha; // Remove a linha da matriz.
00359         }
00360
00361
00362         Lista* elementoAtual = LinhaAtual->linha;
00363         Lista* tempElemento;
00364
00365     }
00366
00367     return matriz; // Retorna o apontador para a matriz atualizada.
00368 }
```

5.6.1.13 somaMaiores()

```
int somaMaiores (
            Matriz * m )
```

Soma os valores maiores da linha.
Percorre todas as linhas da matriz, e identifica o maior de todas as linhas.

ParÃ¢metros

matriz	Apontador para a matriz de qual iremos somar as linhas.

```
00424                           {
00425
00426
00427     Matriz* Linhas = m; // Um apontador para percorrer as linhas da matriz.
00428     int total = 0; // Acumular a soma dos maiores valores encontrados.
00429     int contadorColunas = 0; // Declara uma variável para contar as colunas
00430     int maior = 0;
00431
00432     // Enquanto houver linhas na matriz são percorridas.
00433     while (Linhas != NULL)
00434     {
00435         // Inicializa um apontador 'colunas' para percorrer os elementos (valores) de cada linha.
00436         Lista* colunas = Linhas->linha;
00437         contadorColunas = 0; // Reinicia o contador de colunas
00438         maior = 0; // Inicializa a 0 o maior no inicio de cada linha
00439
00440         // Enquanto houver elementos percorre
00441         while (colunas != NULL)
00442         {
00443             // Condição que derfine o maior da linha
00444             if (colunas->num > maior)
00445             {
00446                 maior = colunas->num; // Determina o maior das linhas
00447                 contadorColunas++; // Conta as colunas
00448             }
00449             colunas = colunas->prox; // Avança de coluna
00450         }
00451
00452         total += maior;// Adiciona o maior valor encontrado na linha
00453
00454         Linhas = Linhas->ProxLinha; // Avança para a proxima linha
00455     }
00456
00457
00458     return total; // Retorna o total acumulado dos maiores valores encontrados em cada linha.
00459 }
```

5.7 ReferÃªncia ao ficheiro C:/Users/hugoc/OneDrive - Instituto PolitÃ©cnico do CÃ¡vado e do Ave/2023_2024/Estruturas de Dados AvanÃ§adas/Dev/Listas ligadas - Fase 1/src/ResoluÃ§Ã£o/main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include "Lib/header.h"
```

FunÃ§Ãµes

- int main ()

  Função principal que utilizamos as funções criadas.

5.7.1 DocumentaÃ§Ã£o das funÃ§Ãµes

5.7.1.1 main()

```
int main ( )
```

Função principal que utilizamos as funções criadas.
Cria uma matriz a partir de dados carregados de um arquivo, modifica valores, insere e remove linhas e colunas, e exibe o estado final da matriz.

```
00016           {
00017
00018     setlocale(LC_ALL, "portuguese"); // Formatação correta de caracteres.
00019
00020     int opcao;
00021     //Definir tamanho da matriz
00022     int Tamlinha, Tamcoluna;
00023     //Define o local a modificar e qual o valor.
00024     int linhaMudar, colunaMudar, valor;
00025     //Onde vamos acrescentar as linhas ou colunas.
00026     int linha, coluna;
00027     int soma;
00028     //Arrays para preencher linhas e colunas adicionadas
00029     int valores[15] = { 344, 32, 23, 61, 83, 324, 12, 43,65,  29, 87, 21, 45, 134, 56 };
00030
00031     // Cria a matriz e define o tamanho
00032     printf("Define o tamanho da matriz (l,c)\nOpção: ");
00033     scanf_s("%d,%d", &Tamlinha, &Tamcoluna);
00034     Matriz* matriz = DistribuirDados(Tamlinha, Tamcoluna);
00035     printf("\n");
00036     MostrarMatriz(matriz);
00037
00038
00039     do {
00040         printf("\nMenu:\n");
00041         printf("1. Mudar valor de uma célula\n");
00042         printf("2. Inserir linha\n");
00043         printf("3. Inserir coluna\n");
00044         printf("4. Remover linha\n");
00045         printf("5. Remover coluna\n");
00046         printf("6. Mostrar matriz\n");
00047         printf("7. Soma\n");
00048         printf("8. Sair\n");
00049         printf("Escolha uma opção: ");
00050         scanf_s("%d", &opcao);
00051
00052         switch (opcao) {
00053         case 1:
00054             //Mudar um valor
00055             printf("\nDefine a posição a mudar e o valor a inserir (l,c,v)\nOpção: ");
00056             scanf_s("%d,%d,%d", &linhaMudar, &colunaMudar, &valor);
00057             MudarValor(matriz, linhaMudar, colunaMudar, valor);
00058             printf("\n");
00059             MostrarMatriz(matriz);
00060             break;
00061         case 2:
00062             // Adicione aqui a lógica para ler linha_abaixo, Tamcoluna, valores
00063             printf("\nDefine a seguir de que linha pretende acrescentar a linha (l)\nOpção: ");
00064             scanf_s("%d", &linha);
00065             matriz = InserirLinha(matriz, linha, Tamcoluna, valores);
00066             printf("\n");
00067             MostrarMatriz(matriz);
```

```
00068             break;
00069         case 3:
00070             // Adicione aqui a lógica para ler coluna_direia, valores
00071             printf("\nDefine a seguir de que coluna pretende acrescentar a coluna (c)\nOpção: ");
00072             scanf_s("%d", &coluna);
00073             matriz = InserirColuna(matriz, coluna, valores);
00074             printf("\n");
00075             MostrarMatriz(matriz);
00076             break;
00077         case 4:
00078             // Remove linha
00079             printf("\nDefine a linha que pretende remover (l)\nOpção: ");
00080             scanf_s("%d", &linha);
00081             matriz = RemoverLinha(matriz, linha);
00082             printf("\n");
00083             MostrarMatriz(matriz);
00084             break;
00085         case 5:
00086             // Remove coluna
00087             printf("\nDefine a coluna que pretende remover (c)\nOpção: ");
00088             scanf_s("%d", &coluna);
00089             matriz = RemoverColuna(matriz, coluna);
00090             printf("\n");
00091             MostrarMatriz(matriz);
00092             break;
00093         case 6:
00094             //Apresenta a matriz
00095             printf("\n");
00096             MostrarMatriz(matriz);
00097             break;
00098         case 7:
00099             // Soma dos maiores valores das linhas
00100             soma = somaMaiores(matriz, Tamlinha, Tamcoluna);
00101             printf("\nSoma dos maiores valores das linhas: %d\n", soma);
00102             break;
00103         case 8:
00104             //Sair
00105             printf("Sair...\n\n");
00106             break;
00107         default:
00108             printf("Opção inválida!\n");
00109         }
00110     } while (opcao != 8);
00111
00112     free(matriz);
00113     return 0; // Termina a execução do programa com sucesso.
00114
00115 }
```

Índice