****

***Curso: Ciência da Computação***

***Disciplina: Estruturas de Dados***

***Profª Luciana Ap. Oliveira Betetto***

**Aula 01**

1. **Algoritmos e a Solução de Problemas**

*Algoritmo*: Qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída (Cormen, 2002).

Todos os problemas a serem resolvidos por algoritmos possuem dados. Estes são armazenados em estruturas, escolhidas de acordo com as operações que podem ser realizadas sobre elas e com o custo de cada uma dessas operações.

Uma linguagem de programação é um método padronizado para expressar instruções para um computador, ou seja, é um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador.

**O que é análise de algoritmos?**

Segundo Cormen (2002), é a previsão dos recursos de que o algoritmo necessitará.

* Memória.
* Largura de banda de comunicação.
* *Hardware* de computação.
* Tempo de computação.

Envolve dois tipos de problemas distintos:

- análise de um algoritmo particular: calcular o custo de um determinado algoritmo na resolução de um problema.

- análise de uma classe de algoritmos: determinar o algoritmo de menor custo possível para resolver um problema.

**Modelo RAM:**

- Instruções executadas uma após a outra, sem operações concorrentes.

- Instruções encontradas em computadores reais, tais como instruções aritméticas, de movimentação de dados e de controle.

Tempo de execução de um algoritmo: função de custo *T*, onde *T*(*n*) é a medida do tempo necessário para executar um algoritmo para um problema de tamanho *n*.

Um algoritmo nem sempre se comporta de modo uniforme, podendo identificar três casos:

*Pior caso* = maior tempo de execução sobre todas as entradas de tamanho *n*.

*Melhor caso* = menor tempo de execução sobre todas as entradas de tamanho *n*.

*Caso médio* = média dos tempos de execução sobre todas as entradas de tamanho *n*.

O tempo de execução de um algoritmo aumenta proporcionalmente ao tamanho da entrada *n* do problema. A escolha do algoritmo ocorre pelo desempenho em entradas grandes e pelo estudo da eficiência*.*

1. **Estruturas de Dados**

Um computador é uma máquina que manipula dados. O estudo da ciência da computação inclui o exame da organização, manipulação e utilização destes dados num computador. Consequentemente, é muito importante entender os conceitos de organização e manipulação de dados.

Nesta grande evolução do mundo computacional, um fator de relevante importância é a forma de armazenar os dados. Então de nada adiantaria o grande desenvolvimento do hardware e do software se a forma de armazenamento e tratamento do dado não acompanhasse esse desenvolvimento. Por isso a importância das estruturas de dados, que nada mais são do que formas otimizadas de armazenamento e tratamento dos dados eletronicamente. Cada tipo de estrutura de dados possui vantagens e desvantagens e cada uma delas tem sua área de atuação otimizada.

A fim de se compreender corretamente o conceito de estrutura de dados e o seu papel no processo de desenvolvimento de software é importante entender e diferenciar o significado entre:

• Tipos de Dados

• Tipos Abstratos de Dados

• Estruturas de Dados

* 1. **Tipos de Dados**

A unidade básica de memória do computador é chamada de bit (binary digit), sendo 0 e 1 usados para representar os dois possíveis estados de um bit particular. Obviamente, com um único bit não podemos representar muita informação. Entretanto, a memória do computador é constituída de uma grande quantidade de bits que, uma vez agrupados e devidamente interpretados, são capazes de representar uma enorme quantidade e variedade de informações.

Os dados manipulados por um algoritmo podem possuir natureza distinta, isto é, podem ser números, letras, frases etc. Dependendo da natureza de um dado, algumas operações podem ou não fazer sentido quando aplicadas a eles. Por exemplo, não faz sentido falar em somar duas letras - algumas linguagens de programação permitem que ocorra a soma dos valores ASCII correspondentes de cada letra.

Num programa, áreas de memória para armazenamento de dados são representados por *variáveis*. A forma como os bits numa variável são agrupados, interpretados e manipulados pelo computador é definida pelo seu tipo de dados. Por exemplo, num programa que calcula o perímetro de uma circunferência a partir do seu raio, precisamos usar variáveis do tipo *real*.

Para poder distinguir dados de naturezas distintas e saber quais operações podem ser realizadas com eles, os algoritmos lidam com o conceito de tipo de dados. O tipo de um dado define o conjunto de valores que uma variável pode assumir, bem como o conjunto de todas as operações que podem atuar sobre qualquer valor daquela variável. Por exemplo, uma variável do tipo inteiro pode assumir o conjunto de todos os números e de todas as operações que podem ser aplicadas a estes números.

**Os tipos de dados devem ser entendidos como sendo o conjunto de valores que uma variável pode assumir ou ainda o conjunto de valores que possam ser gerados por uma função.** Por exemplo, uma variável do tipo *boolean* pode assumir o valor *true* ou *false*. Na maioria dos computadores os tipos fundamentais, também conhecidos como primitivos já fazem parte das características do próprio sistema.

- Os tipos básicos ou primitivos são:

*Inteiro*: compreende o conjunto de todos os números inteiros.

*Real*: representa o conjunto dos números reais.

*Booleano*: pode assumir apenas dois valores True ou False.

*Char*: compreende o conjunto de caracteres alfanuméricos.

- Os estruturados são:

*Sub-Intervalo*: compreende um intervalo com início e fim específico.

*Vetores*: consiste de componentes homogêneos, isto é, do mesmo tipo.

*Registros*: consiste na junção de diferentes tipos de dados em um único tipo, que é o registro.

- Os operadores aritméticos básicos:

*Operadores fundamentais*: ( + ) Adição, ( - ) Subtração, ( \* ) Multiplicação, ( / ) Divisão, que também são considerados fundamentais por representarem as operações aritméticas básicas.

**O tipo de dado é o conceito de como um computador interpreta os bits.**

A maioria das linguagens de programação oferece um conjunto básico de tipos de dados predefinidos (por exemplo, inteiro, real, caractere e lógico), bem como alguns mecanismos que permitem criar tipos de dados a partir de outros predefinidos (por exemplo, vetor, registro e ponteiro).

* 1. **Tipos Abstratos de Dados (TAD)**

Durante o processo de computação de sua saída, um algoritmo manipula dados obtidos de sua entrada. **Quando os dados são dispostos e manipulados de forma homogênea, constituem um *tipo abstrato de dados*.** Tipo Abstrato de Dado (TAD) é um modelo matemático definido por um conjunto de valores e por um conjunto de operadores que atuam sobre esses valores. Quando empregados para a solução de problemas por computador, TAD´s servem tanto para especificar características relevantes das entidades envolvidas nos problemas, quanto para definir de que forma elas se relacionam entre si e como podem ser manipuladas. Uma estrutura de dados é um meio para armazenar e organizar dados com o objetivo de facilitar o acesso e as modificações (Cormen, 2002).

Um exemplo é o de um estudante. Em um projeto anterior à teoria de TAD, um estudante seria representado por variáveis soltas (como seu nome, sua idade e sua matrícula) que seriam operadas separadamente, sem ligação lógica entre elas além do conhecimento do programador de que a variável trata-se do nome da "entidade" estudante. Conceitualmente um programa passou a ser projetado pensando que não há o nome, idade e matrícula do estudante, mas simplesmente o tipo estudante. Este tipo, como um tipo simples (inteiro ou string), deve ter operadores próprios. Assim, o estudante deve possuir operações desejáveis ao programador, como duplicar sua informação, validar a matrícula, verificar a idade, etc. Na prática, o TAD é implementado usando-se um tipo composto (struct/record - estrutura/registro) com os valores pertencentes ao TAD (nome, idade, matrícula) e por funções que operam esta estrutura.

Assim, para usar um TAD na solução de problemas por computador, precisamos antes transformá-lo num *tipo de dados concreto* ou, simplesmente, *tipo de dados*. A transformação de um TAD em um tipo de dados é chamada *implementação*. É durante a implementação que a estrutura de armazenamento dos valores é definida e os algoritmos que efetuam operações nela são projetados.

Tipo de dados concreto

Tipo de dados abstrato

Implementação

Implementação de um tipo de dados

A abstração de informações através do TAD permitiu a melhor compreensão dos algoritmos e maior facilidade de programação, e por consequência aumentou a complexidade dos programas, tornando-se fundamental em qualquer projeto de software a modelagem prévia de seus dados.

Os TAD´s encapsulam a representação dos dados e as operações que podem ser realizadas sobre eles. Na visão do usuário, ele só “enxerga” a interface, não a implementação; não importa se a representação é feita com palitos, números decimais, ou em binário desde que a gente consiga somar, subtrair, multiplicar, etc. Os usuários de um TAD só têm acesso às operações disponibilizadas sobre os dados.

Quando se observa não só como o computador interpreta os bits, mas se observa isso da perspectiva do usuário, isto é, o que os usuários pretendem fazer ou obter a partir desses bits ou tipos, então está ocorrendo uma abstração da realidade, a qual será representada de forma computacional. Um TAD é a abstração em si onde o tipo abstrato é definido e as operações associadas são específicas para a implementação de certo tipo de dado. **Exemplos de TAD: Listas, Pilhas, Filas, Árvores, etc.**

* 1. **Estrutura de Dados (ED)**

Estrutura de dados é qualquer meio utilizado para armazenar e recuperar informações.

É uma forma específica de se implementar ou representar um TAD numa linguagem de programação. Cada ED é construída a partir dos tipos básicos (*int*, *real*, *char...*) e/ou dos tipos estruturados (*array*, *record*) de uma linguagem de programação.

**Exemplo de uma Estrutura de Dados em C:**

Para a criação de novos tipos de estrutura de dados em C utiliza-se a palavra-chave: **typedef**

typedef struct pessoa {

char nome[40];

int idade;

char sexo;

real altura;

};

struct pessoa cliente;

No exemplo acima, criou-se uma estrutura de dados ***pessoa***, onde cada ***pessoa*** possui um ***nome*** (com até 40 caracteres), ***idade***, ***sexo*** e ***altura***. Em seguida, criou-se uma variável ***cliente*** a qual assumirá esses tipos de dados.

Uma lista é a **estrutura de dados** mais básica do **Python** e armazena os **dados** em sequência, onde cada elemento possui sua posição na lista, denominada de índice. O **primeiro elemento** é sempre o **índice zero** e a cada elemento inserido na lista esse valor é incrementado.

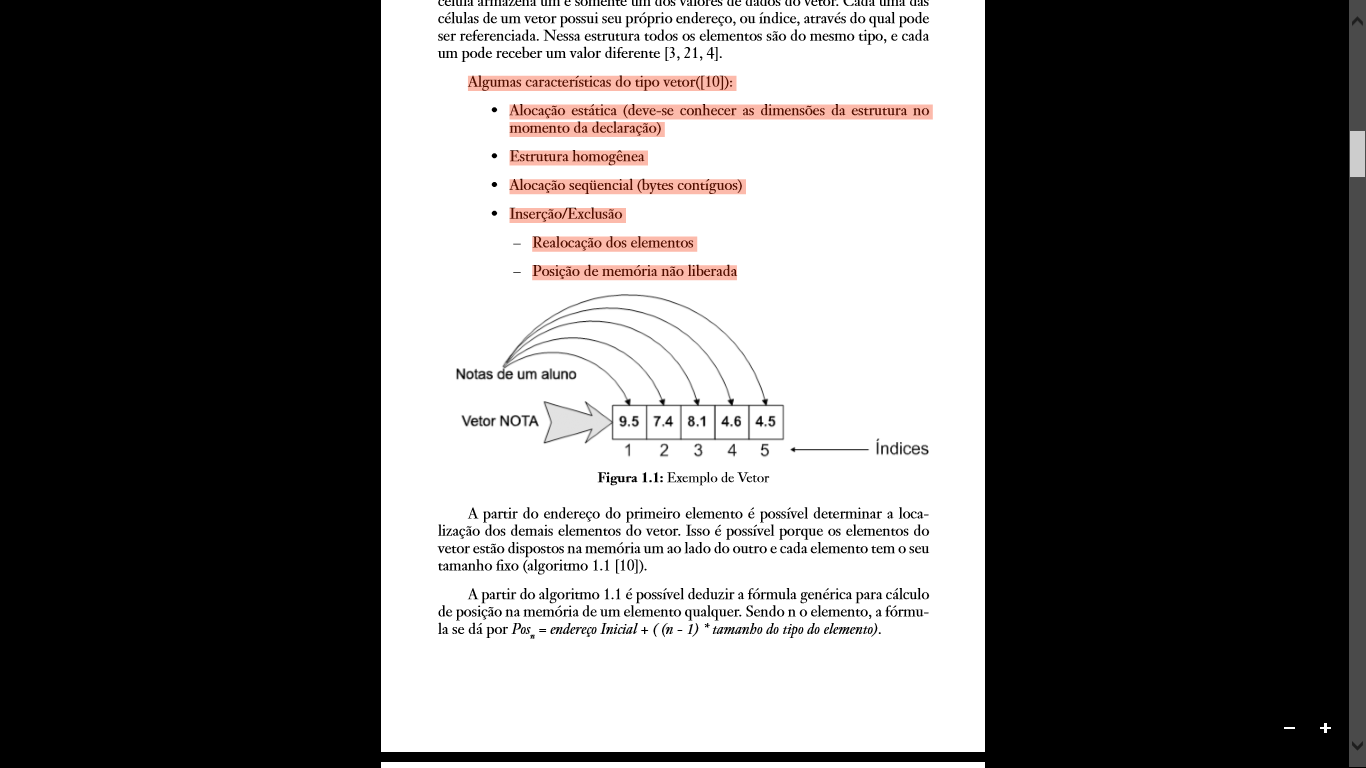
1. **Dados Homogêneos**

Uma estrutura de dados, que utiliza somente um tipo de dado, em sua definição é conhecida como dados homogêneos. Variáveis compostas homogêneas correspondem a posições de memória, identificadas por um mesmo nome, individualizado por índices e cujo conteúdo é composto do mesmo tipo. Sendo os **vetores** (também conhecidos como estruturas de dados unidimensionais) e as **matrizes** (estruturas de dados bidimensionais) os representantes dos dados homogêneos.

* 1. **Vetor**

O vetor é uma estrutura de dados linear que necessita de somente um índice para que seus elementos sejam endereçados. É utilizado para armazenar uma lista de valores do mesmo tipo, ou seja, o tipo vetor permite armazenar mais de um valor em uma mesma variável. Um dado vetor é definido como tendo um número fixo de células idênticas (seu conteúdo é dividido em posições). Cada célula armazena um e somente um dos valores de dados do vetor. Cada uma das células de um vetor possui seu próprio endereço, ou índice, através do qual pode ser referenciada.

Algumas características do tipo vetor([5]):

* Alocação estática (deve-se conhecer as dimensões da estrutura no momento da declaração)
* Estrutura homogênea
* Alocação sequencial (bytes contíguos)
* Inserção/Exclusão
* Realocação dos elementos
* Posição de memória não liberada

Exemplo de uso de vetores em C: Receber 5 valores inteiros e calcular a média.

/\* programa\_vetor.c \*/

#include <stdio.h>

#define TAMANHO 5

int main (void)

{

int iIndice;

int iValorA;

int iSoma;

int aVetor [TAMANHO];

float fMedia;

for (iIndice = 0; iIndice < TAMANHO; iIndice++)

{

printf("Entre com o valor %d : ", iIndice + 1);

scanf("%d", &iValorA);

aVetor[iIndice] = iValorA;

}

iSoma = 0;

for (iIndice=0; iIndice < TAMANHO; iIndice++)

{

iSoma += aVetor[iIndice];

}

fMedia = (float) iSoma/TAMANHO;

printf ("Media : %f\n", fMedia);

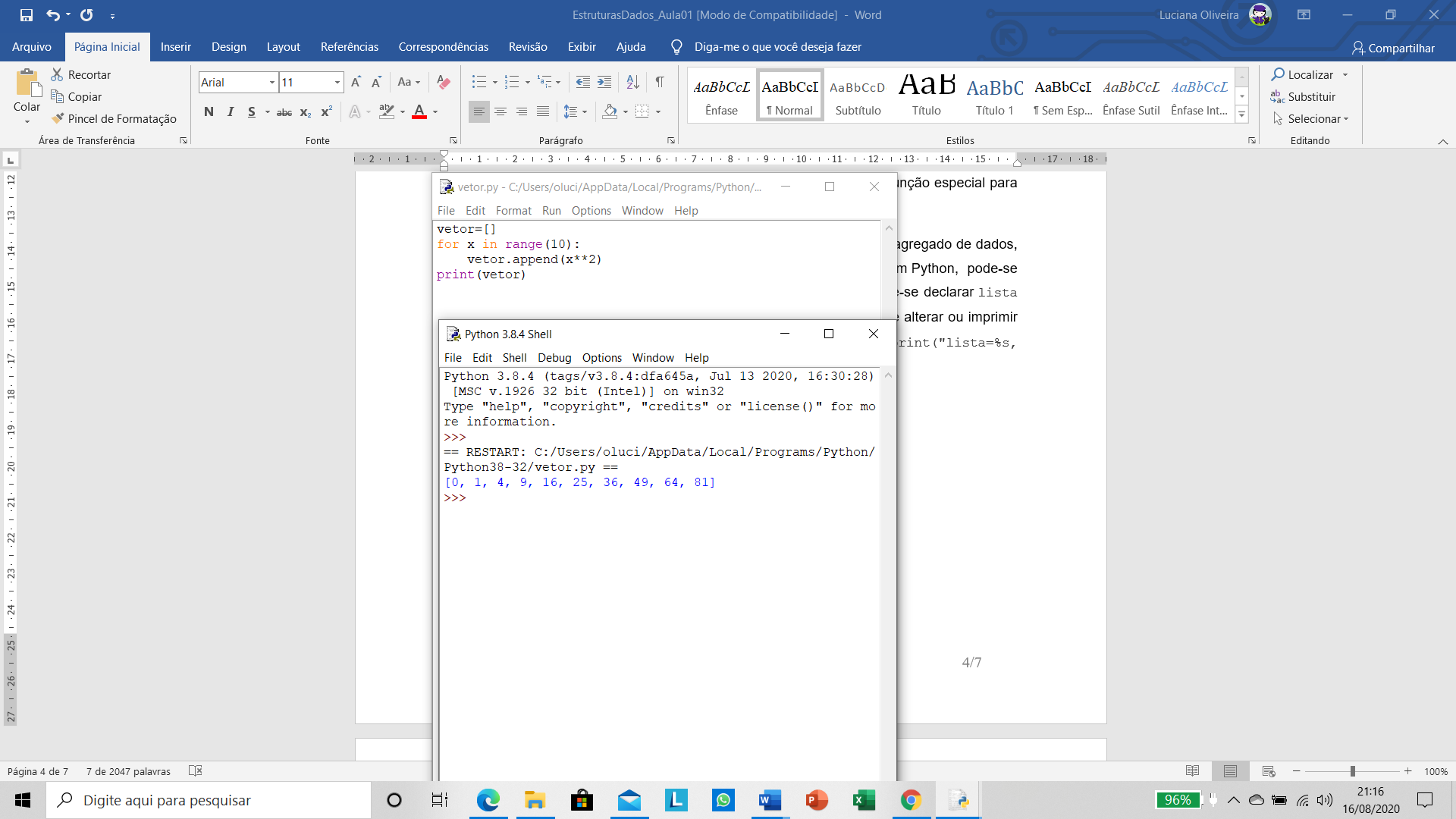
return 0;

}

A linguagem Python encapsula bastante o conceito de vetor, agregando funcionalidades e por isso geralmente o material didático sobre vetores em Python utiliza o nome lista.

Em Python, inserir vários dados de uma só vez não é tão trivial, é preciso processamento para recuperar os valores, para isso é preciso ler a entrada toda como uma cadeia de caracteres (string). Além disso, no Python 2 existe uma função especial para isso, a raw\_input, que no Python 3 sumiu!

Geralmente, na literatura de programação, o termo vetor refere-se a um agregado de dados, de mesmo tipo e em posições consecutivas de memória. Como em Python, pode-se misturar tipos, então usa-se o termo **lista** (ou **tuple**). Por exemplo, pode-se declarar lista = (1, 2, "tres", "VI"); ou tupla = [1, 2, "tres", "VI"]; e alterar ou imprimir um elemento, como em: lista[3] = 3; tupla[3] = 3; print("lista=%s, tupla=%s, lista[3]=%d" % (lista,tupla, lista[3]);.

Exemplo de uso de “vetores” em Python:

Criar vetor com 10 posições:

vetor=[ ]

for x in range(10):

vetor.append(x\*\*2)

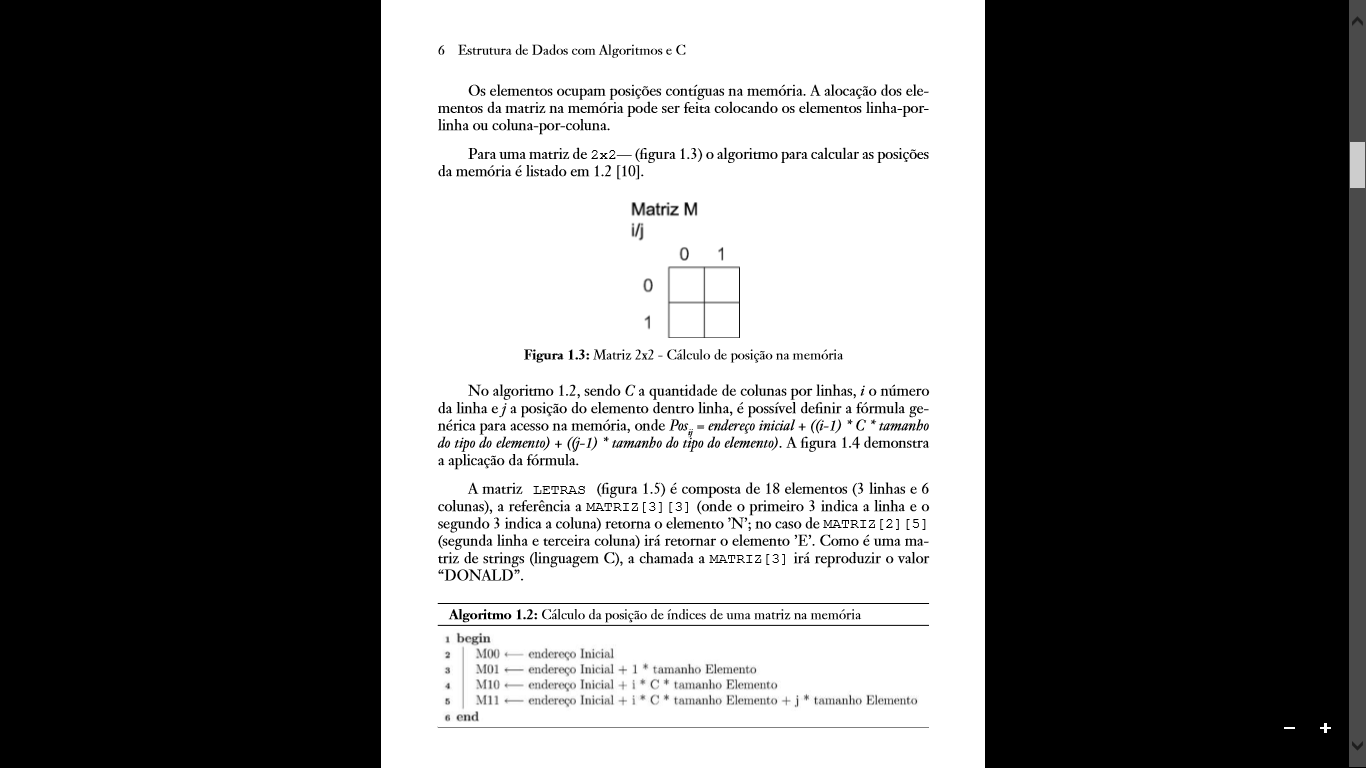
print(vetor)

* 1. **Matriz**

Uma matriz é um arranjo bidimensional ou multidimensional de alocação estática e sequencial. A matriz é uma estrutura de dados que necessita de um índice para referenciar a linha e outro para referenciar a coluna para que seus elementos sejam endereçados. Da mesma forma que um vetor, uma matriz é definida com um tamanho fixo, todos os elementos são do mesmo tipo, cada célula contém somente um valor e os tamanhos dos valores são os mesmos (em C, um char ocupa 1 byte e um int 4 bytes). Os elementos ocupam posições contíguas na memória. A alocação dos elementos da matriz na memória pode ser feita colocando os elementos linha-por-linha ou coluna-por-coluna.

A linguagem C permite ainda trabalhar com matrizes de várias dimensões (matrizes n-dimensionais), embora o seu uso fique mais restrito em aplicações científicas face à sua pouca praticidade de uso.

Exemplo de uso de matrizes em C:

/\* programa\_matriz.c \*/

#include <stdio.h>

#define DIMENSAO 2

int main (void)

{

int iLinha, iColuna;

int iDeterminante;

int iValorA;

int aMatriz [DIMENSAO][DIMENSAO];

/\* Uma regra que se pode sempre levar em consideração: para cada dimensão de uma matriz, sempre haverá um

laço (normalmente um for). Se houver duas dimensões, então haverá dois laços. \*/

for (iLinha=0; iLinha < DIMENSAO; iLinha++)

{

for (iColuna=0; iColuna < DIMENSAO; iColuna++)

{

printf ("Entre item %d %d:", iLinha + 1, iColuna + 1);

scanf ("%d", &iValorA);

aMatriz [iLinha][iColuna] = iValorA;

}

}

iDeterminante = aMatriz[0][0] \* aMatriz [1][1] - aMatriz[0][1] \* aMatriz [1][0];

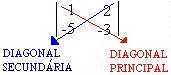
printf ("Determinante : %d\n", iDeterminante);

return 0;

}

***\* Recordando....*** Podemos calcular o determinante de qualquer matriz desde que essa seja quadrada, ou seja, que a matriz tenha o mesmo número de linhas e de colunas (seja uma matriz de ordem n x n).

**Matriz de ordem 2**

Dada a matriz A, de ordem dois A = determinante%20mundo , o seu determinante será calculado da seguinte forma:  
  
O determinante de ordem dois possui uma diagonal principal e uma diagonal secundária.  


O cálculo do seu valor numérico é feito pela diferença do produto da diagonal principal com o produto da diagonal secundária.

determinante%20mundo3

det A = = - 3 – (- 10) = - 3 + 10 = 7

**Referências:**

Ascencio, A.F.G. & Araujo, G.S. Estruturas de Dados. Editora Pearson, 2011.

Cormen, Thomas H. Algoritmos. Editora Campus, 2002.

Laureano, Marcos. Estrutura de Dados com Algoritmos e C. Editora Brasport, 2008.