#### Estruturas Homogêneas

**Introdução**

Ao longo desta disciplina, aprenderemos que uma **ESTRUTURA DE DADOS** é uma **abstração** de objetos do mundo real, que temos interesse de representar na concepção de soluções baseadas em sistemas de informação. Uma **Estrutura de Dados** é composta:

1. Pela estrutura de armazenamento dos **atributos** relevantes do objeto em pauta; e
2. Pelas **operações** sobre estes atributos.

A estrutura de armazenamento representa os **aspectos estáticos** do objeto modelado, ou seja, **o que o objeto é,**  ou seu **estado**. Já as operações representam os **aspectos dinâmicos** do objeto, ou seja, **o que o objeto faz e como é feito,** ou seu **comportamento** .

Numa linguagem estruturada, como em C, os atributos são implementados por meio de **variáveis de tipos primitivos ou estruturados** (**registros** ( ***struct*** ) e **vetores**) enquanto que as operações são implementadas por ***funções***. Já numa linguagem orientada a objetos, como o Java ou Python, **Estruturas de Dados** são implementadas por meio de **classes**, onde os atributos são representados por **variáveis de instância** (de tipos primitivos ou estruturados) e as operações são implementadas por **métodos**.

Deste modo, inicia-se o presente curso buscando entender melhor o conceito de variável e de tipos de dados, primitivos (*built-in*) e estruturados.

Ao declarar uma variável (primitiva ou estruturada), o programador deve indicar:

. seu nome (que será utilizado pelo programador para referenciar a variável em seu programa); e

. seu tipo de dado (primitivo ou estruturado, que indica o conjunto de valores que pode ser armazenado nesta variável).

Um tipo de dado é chamado **primitivo** (ou ***built-in***) se ele já é fornecido nativamente pela linguagem de programação. Obviamente, o conjunto de tipos primitivos varia de tecnologia para tecnologia. Para efeito deste curso, considera-se os seguintes tipos primitivos:

- **int**: valores numéricos inteiros;

- **float**: valores numéricos fracionários ou reais;

- **char**: armazena um caractere (tabela ASCII);

- **string**: armazena uma cadeia de caracteres;

- **boolean**: armazena um valor lógico (V/F).

Já os tipos de dados **estruturados** são basicamente 2:

1. **Estruturas Homogêneas**: são estruturas formadas por dados de mesmo tipo, ou seja, **arranjos** (vetores – uma dimensão e matrizes – duas ou mais dimensões)

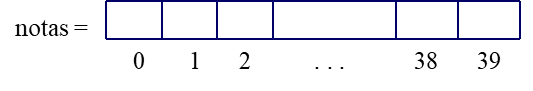
Para declarar um vetor, pode-se usar a seguinte sintaxe:

nome\_var [tamanho] : tipo\_dado\_base

Exemplo:

notas[40]: float

Este exemplo cria um vetor chamado notas, com 40 elementos do tipo float, indexados de 0..39, como abstraído na figura a seguir:

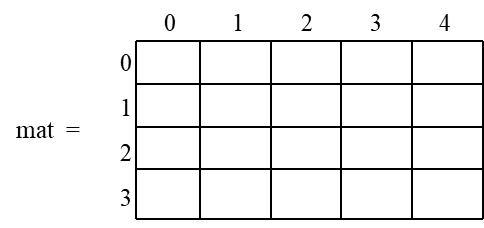


Para declarar uma matriz de duas ou mais dimensões, basta informar dentro dos colchetes o tamanho de cada dimensão, separados por vírgula.

Por exemplo, para declarar uma matriz de nome mat, com dimensão 4x5 (quatro linhas por cinco colunas), armazenando strings, basta declarar:

mat[4, 5] : string

Esta declaração cria uma estrutura abstraída na figura abaixo:



1. **Estruturas Heterogêneas**: permitem agrupar um ou mais elementos de diferentes tipos, primitivos ou estruturados, formando registros. Na próxima atividade, este tipo de dado será mais detalhado e explorado.

**Problemas a serem resolvido**

Para diagnosticar sua habilidade na manipulação de estruturas de dados homogêneos, contrua algoritmos para resolver os seguintes problemas:

1. Algoritmo que leia as notas de uma prova de determinada turma com 50 alunos, calcule a média da turma e mostre na tela as notas maiores que a média e suas respectivas posições no vetor.

**Algoritmo** Media\_Notas

**Constante** Max = 50

**Variáveis** Notas [Max] : **real**

Media, Soma: **real**

k: **inteiro**

## Inicio

Soma 🡨 0

**Para** k **de** 1 **ate** Max **faca**

**Escreva** ( “Informe a nota do aluno “, k )

**Leia** ( Notas[k] )

Soma 🡨 Soma + Notas[k]

**Fimpara**

Media 🡨 Soma / Max

**Para** k **de** 1 **ate** Max **faca**

**Se** (Notas[k] > Media) **entao**

**Escreval** (“Aluno “, k , “ com nota maior que a média da turma “ )

**Fimse**

**Fimpara**

**FimAlgoritmo**

1. Algoritmo que leia um conjunto X com N elementos reais e calcule a diferença entre o maior e o menor elemento existente, bem como as posições que os mesmos ocupam no conjunto. Esta diferença entre o maior e menor elementos de um conjunto de valores é denominada *amplitude*.

**Algoritmo** Amplitude

**Constante** N = 100

**Variáveis** X [N] : **real**

i, posMaior, posMenor : **inteiro**

## Inicio

***{ Dá carga no vetor X }***

**Para** i **de** 0 **ate** N-1 **faca**

**Escreva** ( “Informe um valor qualquer “ )

**Leia** ( X [i] )

**Fim para**

***{ Pesquisa a posição do menor e maior elemento de X }***

posMaior 🡨 0

posMenor 🡨 0

**Para** i **de** 1 **ate** N - 1 **faca**

**Se** ( X[ i ] > X[posMaior]) **Entao**

posMaior 🡨 i

**Fimse**

**Se** ( X[ i ] < X[posMenor]) **Entao**

posMenor 🡨 i

**Fimse**

**Fimpara**

***{ Mostra amplitude e posição do maior e menor valores }***

**Escreval** ( “Amplitude = “, X[posMaior] – X[posMenor] )

**Escreval** ( “Posição do maior elemento: “, posMaior )

**Escreval** ( “Posição do menor elemento: “, posMenor )

**FimAlgoritmo**

1. Algoritmo que leia um vetor X[20] e após, troque o 1o elemento pelo 20o, o 2o pelo 19o, e assim por diante. Ao final, imprima X.

**Algoritmo** Troca

**Constante** max = 20

**Variáveis** X [max] : **inteiro**

i , aux : **inteiro**

## Inicio

***{ Dá carga no vetor X }***

**Para** i **de** 0 **Ate** max-1 **faca**

**Escreva** ( “Informe um valor inteiro qualquer “ )

**Leia** ( X [ i ] )

**FimPara**

***{ Inverte valores – observe que o laço vai apenas até metade do vetor }***

**Para** i **de** 0 **Ate** (Max **div** 2 - 1) **faca**

aux 🡨 X [ i ]

X [ i ] 🡨 X [ max – i – 1 ]

X [ max – i – 1 ] 🡨 aux

**FimPara**

***{ Mostra vetor invertido }***

**Para** i **de** 0 **Ate** Max-1 **faca**

**Escreval** ( X [ i ] )

**FimPara**

**FimAlgoritmo**

1. Algoritmo que, considerando uma matriz 20x10 já carregada com valores inteiros, efetue a soma de cada linha e escreva as somas obtidas. Mostrar também os números das linhas que apresentarem a maior e menor soma.

**Algoritmo** Soma\_Linhas

**Variáveis** M [ 20, 10] : **inteiro**

Soma [20] : **inteiro**

i, j, posMaior, posMenor : **inteiro**

## Inicio

posMaior 🡨 0

posMenor 🡨 0

***{ Soma linhas e verifica maior e menor }***

**Para** i **de** 0 **até** 19 **faca**

Soma [ i ] 🡨 0

**Para** j **de** 0 **até** 9 **faca**

Soma [ i ] 🡨 Soma [ i ] + M [ i, j ]

**FimPara**

**Se** (Soma [ i ] > Soma [posMaior]) **entao**

posMaior 🡨 i

**Senao**

**Se** (Soma [ i ] < Soma [posMenor]) **entao**

posMenor 🡨 i

**FimSe**

**FimSe**

**FimPara**

***{ Mostra a soma das linhas e a maior e menor soma }***

**Para** i **de** 0 **até** 19 **faca**

**Escreval** ( “Soma da linha “, i , “ = “, Soma [ i ])

**FimPara**

**Escreval** ( “Maior soma = “, Soma [ posMaior], “ na linha “, posMaior )

**Escreval** ( “Menor soma = “, Soma [ posMenor], “ na linha “, posMenor)

**FimAlgorimto**

1. Algoritmo para gerar a seguinte matriz M:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 | 02 | 03 | 04 | 05 |
| 06 | 07 | 08 | 09 | 10 |
| 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 |

**Algoritmo** Gera\_Matriz

**Variáveis** M [ 4, 5 ] : **inteiro**

lin, col : **inteiro**

## Inicio

***{ Carrega a matriz M }***

**Para** lin **de** 0 **até** 3 **faca**

**Para** col **de** 0 **até** 4 **faca**

M [ lin, col ] 🡨 lin \* 5 + col +1

**FimPara**

**FimPara**

***{ Mostra a matriz R }***

**Para** lin **de** 0 **até** 3 **faca**

**Para** col **de** 0 **até** 4 **faca**

**Escreva** ( M [ lin, col ], “ , “ )

**FimPara**

**Escreval**

**FimPara**

**FimAlgoritmo**

1. Algoritmo que, dada uma matriz quadrada de ordem 10, verifique se esta é simétrica. Uma matriz M é simétrica se M[i, j] = M[j, i].

**Algoritmo** “Verifica\_simetria”

**Constante** n = 10

**Variáveis** M [n, n] : **inteiro**

i, j : **inteiro**

achou\_diferente : **logico**

## Inicio

***{ admitido matriz já carregada - pesquisa só na metade abaixo diagonal***

***principal para verificar simetria }***

achou\_diferente 🡨 **falso**

i 🡨 1

**Enquanto** ( i < n) **e** ( **não** achou\_diferente ) **faca**

j 🡨 0

**Enquanto** ( j < i ) **e** ( **não** achou\_diferente ) **faca**

**Se** ( M [ i, j ] <> M [ j, i ] ) **entao**

achou\_diferente 🡨 **verdadeiro**

**Senao**

j 🡨 j + 1

**FimSe**

**Fim enquanto**

i 🡨 i + 1

**Fim enquanto**

**Se** ( achou\_diferente ) **entao**

**Escreval** ( “ Matriz não é simétrica “ )

**Senao**

**Escreval** ( “ Matriz é simétrica “ )

**FimSe**

**FimAlgoritmo**

**Desafio!**

**(*atividade para aula de 21/02*)**

A seguir alguns problemas cuja soluções aplicam indução matemática. Você está desafiado a resolver estas questões!

* 1. Dada uma matriz quadrada **N**x**N**, preenchida com valores inteiros, elabore trechos de código para acessar e somar os elementos indicados, usando indução em função do tamanho **N** e dos índices **lin** e **col**:

1. elementos abaixo da diagonal principal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 |

Soma 🡨 0

**Para** lin **de** 1 **até** N–1 **faca**

**Para** col **de** 0 **até** lin–1 **faca**

Soma 🡨 Soma + M [ lin, col ]

**FimPara**

**FimPara**

**EscrevaL** (Soma)

1. elementos acima da diagonal principal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 |

Soma 🡨 0

**Para** lin **de** 0 **até** N–2 **faca**

**Para** col **de** lin+1 **até** N–1 **faca**

Soma 🡨 Soma + M [ lin, col ]

**FimPara**

**FimPara**

**EscrevaL** (Soma)

1. elementos abaixo da diagonal secundária

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 |

Soma 🡨 0

**Para** lin **de** 1 **até** N–1 **faca**

**Para** col **de** N–lin **até** N–1 **faca**

Soma 🡨 Soma + M [ lin, col ]

**FimPara**

**FimPara**

**EscrevaL** (Soma)

1. elementos acima da diagonal secundária

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 |

Soma 🡨 0

**Para** lin **de** 0 **até** N–2 **faca**

**Para** col **de** 0 **até** N–2–lin **faca**

Soma 🡨 Soma + M [ lin, col ]

**FimPara**

**FimPara**

**EscrevaL** (Soma)

* 1. Ainda aplicando indução, agora em **Séries Matemáticas** e **Somatórios**, elabore trechos de código para resolver os seguintes problemas:

1. Gerar os N primeiros termos da série {1, 4, 9, 16, 25, ...}

O padrão desta série é: X1 = 1; Xk+1 = Xk + 2\*k+1; para k = 1, 2, 3, ...

Portanto temos o algoritmo correspondente (onde se supõe que todas as váriáveis foram previamente declaradas):

**Escreval** (“Digite a quantidade de termos”)

**Leia** (N)

X 🡨 1

**Escreva** (X) **{escreve primeiro termo}**

**Para** k **de** 1 **ate** N-1 **faca**

X 🡨 X + 2\*k + 1 **{calcula termo subsequente}**

**Escreva** (“, ”, X) **{escreve termo subsequente}**

**Fimpara**

1. Gerar os N primeiros termos da série {1, 2, 4, 7, 11, ... }

O padrão desta série é: X1 = 1; Xk+1 = Xk + k; para k = 1, 2, 3, ...

Portanto temos o algoritmo correspondente (onde se supõe que todas as váriáveis foram previamente declaradas):

**Escreval** (“Digite a quantidade de termos”)

**Leia** (N)

X 🡨 1

**Escreva** (X) **{escreve primeiro termo}**

**Para** k **de** 1 **ate** N-1 **faca**

X 🡨 X + k **{calcula termo subsequente}**

**Escreva** (“, ”, X) **{escreve termo subsequente}**

**Fimpara**

1. Gerar os N primeiros termos da série {8, 10, 16, 18, 32, 34, ... }

O padrão desta série é: X1 = 8; Xk+1 = Xk + 2; Xk+2 = Xk \* 2; para k = 1, 3, 5, 7, ...

Portanto temos o algoritmo correspondente (onde se supõe que todas as váriáveis foram previamente declaradas):

**Escreval** (“Digite a quantidade de termos: ”)

**Leia** (N)

Termo 🡨 8

**Escreval** (Termo) **{escreve primeiro termo}**

i 🡨 1

**Enquanto** (i <= N ) **Faça**

**Escreval** (Termo + 2) **{escreve termo par subsequente}**

Termo 🡨 Termo \* 2 **{calcula termo impar subsequente}**

**Escreval** (Termo) **{escreve termo impar subsequente}**

i 🡨 i + 2 **{cada iteração gera dois termos}**

**Fim** **Para**

1. Gerar a soma do N primeiros termos do somatórios:
   1. 

**Escreval** (“Digite a quantidade de termos: ”)

**Leia** (N)

S 🡨 0

**Para** i **de** 1 **até** N **Faça**

S 🡨 S + 1 / i

**FimPara**

**Escreval**(“S = “, S)

* 1. 

**Escreval** (“Digite a quantidade de termos: ”)

**Leia** (N)

S🡨 0

**Para** Den **de** 1 **ate** N **Faça**

S 🡨 S + (( Den \* 2 - 1) / Den )

**FimPara**

**Escreval** (“ S = “ , S)

* 1. 

**Escreval** (“Digite a quantidade de termos: ”)

**Leia** (N)

S 🡨 0

Sinal 🡨1

denom 🡨 1

**Para** i de 1 até N **Faça**

S 🡨 S + Sinal / ( denom \*\* 3 )

Sinal 🡨 Sinal \* (- 1)

denom 🡨 denom + 2

**FimPara**

**Escreval** (“ S = “ , S)

* 1. ... e uma questão para entrar no espírito da disciplina ED: escreva um algoritmo que insira um dado valor em um vetor ordenado ascendente, de modo que este continue ordenado. Por exemplo, ao inserir o valor **9** no vetor V = { 2, 5, 8, 11, 17 }, este deverá ficar V = { 2, 5, 8, **9**, 11, 17 }

**Algoritmo** Insere\_Ordenado

**Constante** max = 100

**Variáveis** V[max]: **inteiro**

Vr\_Ins, i, j, ult: **inteiro**

**Inicio**

***{ Admitindo vetor V já previamente carregado }***

***{ So insere se ainda tiver espaço no vetor, ou seja, ult < max }***

***{ ult marca a posição do último elemento inserido no vetor***

**Se** ( ult < max ) **Entao**

***{ Lê o valor a ser inserido }***

**Escreva** ( “Informe um valor a ser inserido “ )

**Leia** ( Vr\_Ins )

***{ Pesquisa o local de insterção }***

i 🡨 1

**Enquanto** (i <= ult) **e** ( Vr\_Ins > V [ i ] ) **faca**

i 🡨 i + 1

**FimEnquanto**

***{ Desloca todos os elementos desde a posição de inserção até último }***

j 🡨 ult

**Enquanto** ( j > i ) **faca**

V [ j+1] 🡨 V[ j ]

j 🡨 j – 1

**FimEnquanto**

***{ Insere o valor na posição correta e atualiza ult }***

V [ i ] 🡨 Vr\_Ins

ult 🡨 ult +1

**Fimse**

**FimAlgoritmo**