INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ESTRUTURA DE DADOS

TIPO ABSTRATO DE DADO (TAD)

PROFESSOR RESPONSÁVEL:
José Arnaldo Mascagni de Holanda
CONTATOS: arnaldomh@ifsp.edu.br

ALGORITMOS, ESTRUTURA DE DADOS E PROGRAMAS

Algoritmo

...sequencia de ações executáveis para a obtenção de uma solução para um determinado tipo de problema.

(Ziviani, 2003)

Estrutura de Dados

...organização de dados e operações (algoritmos) que podem ser aplicadas sobre esses como forma de apoio à solução de problemas (complexos). *Struct.*

Programas

...formulações concretas de algoritmos abstratos, baseados em representações e estruturas específicas de dados.

(Wirth, 1976)

– algoritmos que podem ser executados em computadores.

TIPOS DE DADOS - DEFINIÇÃO

...**conjunto de valores** a que uma constante pertence, ou que podem ser assumidos por uma variável ou expressão, ou que podem ser gerados por uma função.

(Wirth, 1976)

- Tipos simples ou primitivos: int, float, double etc
- Tipos compostos homogêneos: matrizes
- Tipos compostos heterogêneos ou estruturados: structs

TIPOS DE DADOS - PERSPECTIVAS

- **Máquina**: formas de se interpretar o conteúdo da memória.

- **Usuário/desenvolvedor**: o que pode ser feito com determinado tipo de dado, sem se preocupar como isso é feito em baixo nível.

PROBLEMA: COMO DEFINIR UM NÚMERO RACIONAL?

Possibilidades:

1. Vetor de 2 elementos inteiros: numerador e denominador.
2. Struct com 2 campos inteiros: numerador e denominador.
3. Outras...

DIFERENTES IMPLEMENTAÇÕES PARA O MESMO TIPO DE DADO

- Há diferentes possibilidades de implementações que proporcionam a otimização do código
  - velocidade,
  - economicidade de espaço,
  - clareza na implementação
  - etc

- Para cada uma das implementações, há algo semelhante: os tipos de operações possíveis. Ex: racionais:
  - criar, somar, multiplicar, imprimir, comparar igualdade etc.

MANUTENÇÃO DE CÓDIGOS

Mudanças nas implementações podem gerar impactos para os usuários finais dos programas, eventualmente ocasionando **erros** e até mesmo a **re-implementação** do código.

QUESTÃO PRINCIPAL

Como modificar as implementações dos tipos com o menor impacto possível para os programas que o usam?

Resposta: esconder (“**encapsular**”) a forma de implementação de determinado tipo de dado de quem faz uso dele.

TIPO ABSTRATO DE DADOS (TAD) - DEFINIÇÕES

- ...modelo matemático, acompanhado das operações definidas sobre o modelo. *(Ziviane, 2003)*

- .... novo tipo de dado e o conjunto de operações para manipular dados desse tipo. *(Rangel e Celes, 2002)*

- ... coleção bem definida de dados e um conjunto de operadores que podem ser aplicado em tais dados.

TAD – UTILIDADE E CARACTERÍSTICAS

- Encapsulamento de tipos de dados: pensar em termos de operações suportadas e não como são implementadas;
- Segurança: usuário não conhece a implementação interna de um tipo de dado, não acessa os dados deretamente;
- Flexibilidade de Reuso: é possível alterar um TAD, sem alterar as aplicações que o utilizam.

TAD – MODELO MATEMÁTICO

Resumindo TAD...

Dados/Valores + Operações/funções

- Tupla (v, o), onde:
  - v = conjunto de valores
  - o = conjunto de operações aplicáveis sobre v

- Exemplo:
  - $v = \mathbb{R}$
  - $o = \{+, -, *, /, ==, !=, <, >, <=, >=\}$

TAD – EXEMPLOS

Mundo Real	Dados/Valores	Tipo de Dado	Operações
pessoa 	idade	int	**nasce** (idade <- 0) **aniversário** (idade <- idade + 1;)
cadastro de funcionários 	nome cargo salario	lista ordenada	**entra na lista** **sai da lista** **altera cargo** **altera salario**
fila de espera 	nome posição na fila	fila	**entra na fila** (no fim) **sai da fila** (o primeiro)
pilha de cartas 	naipe valor posição na pilha	pilha	**põe carta na pilha** (topo) **retira carta da pilha** (topo)

TAD – IMPLEMENTAÇÃO

Após definir um TAD e operações associadas, é possível implementá-lo usando uma linguagem de programação.

- É possível fazer diversas implementações para um mesmo TAD, com vantagens e desvantagens.

EXEMPLO DE TAD – MANIPULAÇÃO DE ARQUIVOS EM C

```
FILE* f;
```

→ Os dados de "f" só podem ser acessados pelas funções:

- fopen();
- fclose();
- fputc();
- fgetc();
- feof();
- ...

TAD NA PRÁTICA – MODULARIZAÇÃO (CONVENÇÃO DA LINGUAGEM C)

- Arquivo ".h": interface de módulo
  - Protótipos das funções;
  - Tipos de ponteiros;
  - Dados globalmente acessíveis.

- Arquivo1".c": TAD
  - Declaração do tipo de dados;
  - Implementação das Funções.
- main".c": cliente
  - Usa as funções do TAD.

TAD - EXEMPLO 1: PONTO (X,Y)

```
struct ponto{
    float x;
    float y;
};
```

- 1º passo: definir o arquivo ".h" contento: protótipos; tipos de ponteiro, dados globais.

INTERFACE DE MÓDULO

ponto.h

ponto.c | ponto.h

```
1
2   typedef struct ponto Ponto; //apelido para minha struct
3
4   //Protótipos
5
6   //Cria um novo ponto e devolve um ponteiro para o ponto
7   Ponto* criaPto (float x, float y);
8
9   //Libera um novo ponto
10  Ponto* liberaPto (Ponto* p);
11
12  //Acessa valores de x e y
13  Ponto* acessaPto (Ponto* p, float* x, float* y);
14
15  //Atribui valores de x e y ao ponto p
16  Ponto* atribuiPto (Ponto* p, float x, float y);
17
18  //Calcula distância entre dois pontos
19  Ponto* distanciaPto (Ponto* p1, Ponto* p2);
20
```

TAD - EXEMPLO 1: PONTO (X,Y)

- 2º passo: criar o arquivo ".c" contento: declaração dos tipos e funções de manipulação.

TAD

struct ponto
e
operações

ponto.h [*] ponto.c

```c
#include <stdlib.h>  //função para calculo de distância
#include <math.h> //tem a constante NULL - alocação dinâmica
#include "ponto.h" //inclui minha biblioteca com os protótipos

//definição do tipo de dados
struct ponto{
    float x;
    float y;
};
//criação das funções
Ponto* criaPto (float x, float y){
    //aloca espaço para x e y
    Ponto* p = (Ponto*) malloc(sizeof(Ponto));
    if(p != NULL){
        p->x = x;
        p->y = y;
    }
    return p;
}
//apaga/libera o ponto
void liberaPto (Ponto* p){
    free(p);
}
```

ponto.c

TAD - EXEMPLO 1: PONTO A PARTIR DE (X,Y)

- 2º passo: ...continuação.

TAD

...continuação

ponto.c

ponto.h | ponto.c

```
24  //recupera o valor do ponto por referência.
25  void acessaPto (Ponto* p, float* x, float* y){
26      *x = p->x;
27      *y = p->y;
28  }
29  //atribui valor ao ponto
30  void atribuiPto (Ponto* p, float x, float y){
31      p->x = x;
32      p->y = y;
33  }
34  //cálculo da distânci entre 2 pontos
35  float distanciaPto (Ponto* p1, Ponto* p2){
36      float distX = p1->x - p2->x;
37      float distY = p1->y - p2->y;
38      return sqrt(distX*distX) + (distY*distY);
39  }
```

TAD - EXEMPLO 1: PONTO A PARTIR DE (X,Y)

```
struct ponto{
    float x;
    float y;
};
```

- 3º passo: cria programa *cliente* para acessar a interface e arquivo com operações.

ponto.h | ponto.c | pontoMain.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "ponto.h"
4
5  main(){
6      float dist;
7      //Ponto p; ==> ERRO!
8      Ponto *p, *q;
9
10     //p->x = 10;//ERRO!
11     p = criaPto(10,21);
12     q = criaPto(7,25);
13
14     dist = distanciaPto(p,q);
15
16     printf("Distancia entre pontos: %f\n", dist);
17
18     liberaPto(p); liberaPto(q);
19
20     system("PAUSE");
21 }
```

main.c

REUSO DE BIBLIOTECAS

- Bibliotecas podem ser reusadas a partir do compartilhamento da interface de módulo (arquivo .h) e do compartilhamento do programa-objeto criado (arquivo .o).

- No exemplo anterior, compartilham-se os arquivos:
  - ponto.h e ponto.o (*linkedição*) para reuso em outros programas clientes (*do tipo main*).

O arquivo que contém o TAD e as funções (arquivo.c) não é compartilhado!, preservando, portanto, sua biblioteca.

PROCESSO DE "LINKEDIÇÃO" NO DEVC++

- Crie um novo projeto
  - importe arquivo.h (exibe os protótipos existentes)
  - importe arquivo.o (OBSERVE QUE NÃO FOI NECESSÁRIO TRAZER O ARQUIVO.C)

- Crie a relação de "linkEdição" ao arquivo.o
  - Project
    - Project Options
      - Parameters
        - Add Library or Object
          - → Selecione o arquivo objeto (ponto.o)
            - OK.

Adicione o local da Biblioteca no diretório de include!

- **Crie um novo *main.c* que utilizará a biblioteca "linkEditada".**

EXEMPLO 2: MANIPULAÇÃO DE MATRIZ DINÂMICA REPRESENTADA POR VETOR SIMPLES

matriz.h | main.c | matriz.c

```
1
2   /* Tipo Exportado */
3   typedef struct matriz Matriz;
4
5   /* Funções Exportadas */
6   /* Função cria -Aloca e retorna matriz m por n */
7   Matriz* cria (int m, int n);
8
9   /* Função libera -Libera a memória de uma matriz */
10  void libera (Matriz* mat);
11
12  /* Função acessa -Retorna o valor do elemento [i][j] */
13  float acessa (Matriz* mat, int i, int j);
14
15  /* Função atribui -Atribui valor ao elemento [i][j] */
16  void atribui (Matriz* mat, int i, int j, float v);
17
18  /* Função linhas -Retorna o no. de linhas da matriz */
19  int linhas (Matriz* mat);
20
21  /* Função colunas -Retorna o no. de colunas da matriz */
22  int colunas (Matriz* mat);
```

INTERFACE DE MÓDULO

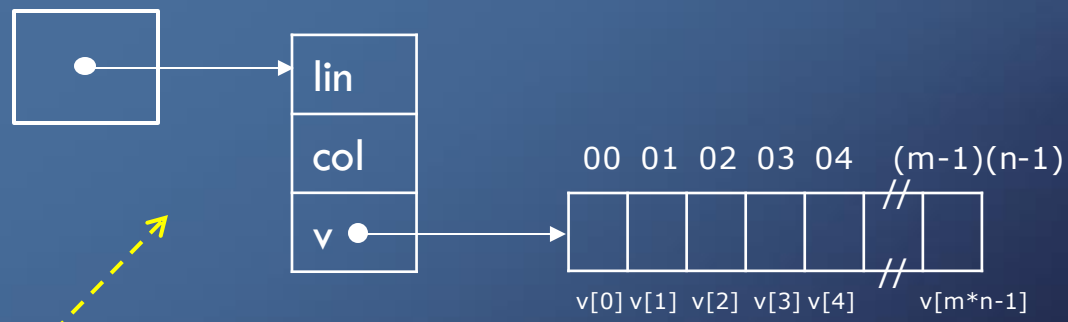
matriz.h

EXEMPLO 2: MANIPULAÇÃO DE MATRIZ DINÂMICA REPRESENTADA POR VETOR SIMPLES

matriz.h | main.c | matriz.c

```c
#include <stdlib.h>/* printf*/
#include <stdio.h>/* malloc, free, exit */
#include "matriz.h"

/* TAD: Matriz m por n (m e n >= 1)*/
struct matriz {
    int lin;
    int col;
    float* v;
};

void libera (Matriz* mat){
    free(mat->v);
    free(mat);
}

Matriz* cria (int m, int n) {
    Matriz* mat= (Matriz*) malloc(sizeof(Matriz));
    if(mat == NULL) {
        printf("Memoria insuficiente!\n");
        exit(1);
    }
    mat->lin = m;
    mat->col = n;
    mat->v = (float*) malloc(m*n*sizeof(float));
```

TAD



matriz.c

EXEMPLO 2: MANIPULAÇÃO DE MATRIZ DINÂMICA REPRESENTADA POR VETOR SIMPLES

```c
float acessa (Matriz* mat, int i, int j) {
    int k; /* índice do elemento no vetor -armazenamento por linha*/
    if (i<0 ||i>=mat->lin||j<0 ||j>=mat->col) {
        printf("Acesso invalido!\n");
        exit(1);
    }
    //k = (i)*mat->col+ j -1; --> ERRO
    k = i * (mat->col) + j;
    printf("v[%d]:%4.2f \n",k, mat->v[k]);
    return mat->v[k];
}

int linhas (Matriz* mat) {
    return mat->lin;
}

void atribui (Matriz* mat, int i, int j, float v) {
    int k; /* índice do elemento no vetor */

    if(i<0 || i>=mat->lin || j<0 || j>=mat->col) {
        printf("Atribuicao invalida!\n");
        exit(1);
    }
    //k = (i)*mat->col + j - 1; --> ERRO
    k = i * (mat->col) + j;
    printf("Inserindo posicao v[%d].... \n",k);
    mat->v[k] = v;
}

int colunas (Matriz* mat) {
    return mat->col;
}
```

TAD

continuação...

matriz.c

EXEMPLO 2: MANIPULAÇÃO DE MATRIZ DINÂMICA REPRESENTADA POR VETOR SIMPLES

matriz.h | main.c | matriz.c

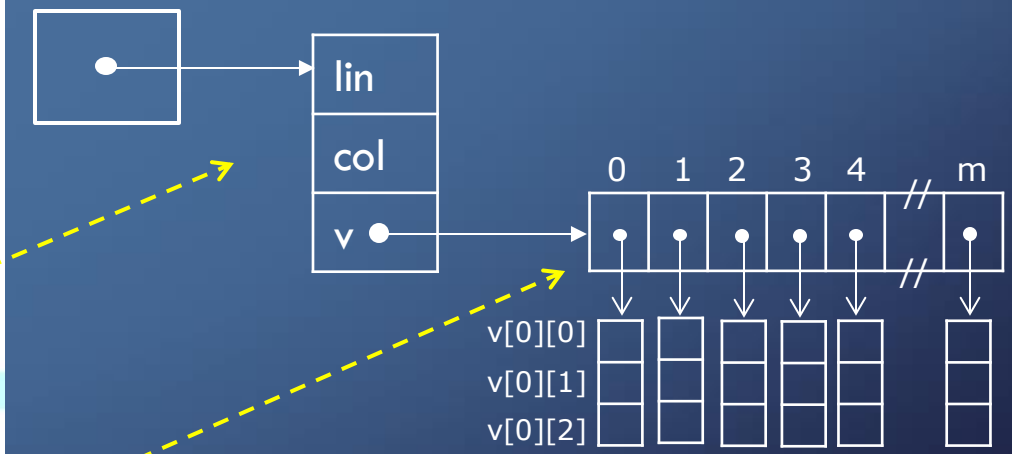
```c
#include <stdio.h>
#include <stdlib.h>
#include "matriz.h"
main()
{
    float a,b,c,d;
    Matriz *m;

    // criação de uma matriz
    m = cria(5,5);

    // inserção de valores na matriz
    atribui(m,0,1,47);
    atribui(m,1,2,13);
    atribui(m,2,4,75);
    atribui(m,4,0,21);
    printf ("\n");

    // verificando se a inserção foi feita corretamente
    a = acessa(m,0,1);
    b = acessa(m,1,2);
    c = acessa(m,2,4);
    d = acessa(m,4,0);

    printf ("\nm[0][1]: %4.2f \n", a);
    printf ("m[1][2]: %4.2f \n", b);
    printf ("m[2][4]: %4.2f \n", c);
    printf ("m[4][0]: %4.2f \n", d);

    system("PAUSE");
}
```

main.c

EXEMPLO 3: MANIPULAÇÃO DE MATRIZ DINÂMICA REPRESENTADA POR PONTEIRO

main.c | matriz.c | matriz.h

```c
#include <stdlib.h>/* printf*/
#include <stdio.h>/* malloc, free, exit */
#include "matriz.h"

/* TAD: Matriz m por n */
struct matriz {
    int lin;
    int col;
    float** v; //PONTEIRO PARA PONTEIRO
};


Matriz* cria (int m, int n) {
    int i;
    Matriz* mat= (Matriz*) malloc(sizeof(Matriz)); //aloca mat
    if(mat == NULL) {
        printf("Overflow!\n");
        exit(1);
    }
    mat->lin = m;
    mat->col = n;
    mat->v = (float**) malloc(m*sizeof(float*)); //ALOCA PARA m PONTEIROS (lin)
    for (i=0; i<m; i++)
        mat->v[i] = (float*) malloc(n*sizeof(float));
                    //PARA CADA POSIÇÃO DE v, ALOCA VETOR PARA n float (col)

    return mat;
}
```

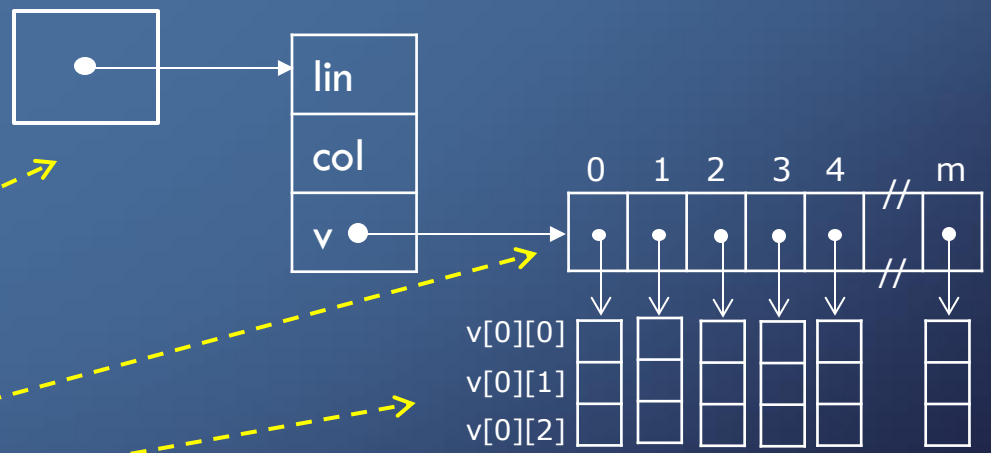


matriz.c

EXEMPLO 3: MANIPULAÇÃO DE MATRIZ DINÂMICA REPRESENTADA POR PONTEIRO

main.c | matriz.c | matriz.h

```c
float acessa (Matriz* mat, int i, int j) {
    int k; /* índice do elemento no vetor -armazenamento por linha*/
    if (i<0 ||i>=mat->lin||j<0 ||j>=mat->col) {
        printf("Acesso invalido!\n");
        exit(1);
    }
    return mat->v[i][j];
}

void atribui (Matriz* mat, int i, int j, float v) {
    int k; /* índice do elemento no vetor */

    if(i<0 || i>=mat->lin || j<0 || j>=mat->col) {
        printf("Atribuicao invalida!\n");
        exit(1);
    }
    mat->v[i][j] = v;
}

void libera (Matriz* mat){
    int i;
    for (i=0; i<mat->lin; i++)
        free(mat->v[i]);
    free(mat->v);
    free(mat);
}

int linhas (Matriz* mat) {
    return mat->lin;
}

int colunas (Matriz* mat) {
    return mat->col;
}
```



matriz.c

DÚVIDAS?

EXERCÍCIOS (TAD)

DESAFIO. VIDE ORIENTAÇÕES: Crie um Tipo Abstrato de Dados (TAD) que represente os números racionais e que contenha as seguintes funções:
(a) Cria racional;
(b) Soma racionais;
(c) Multiplica racionais;
(d) Testa se são iguais;

2. Sabe-se que um número complexo é escrito da forma $x + iy$, onde $i^2 = -1$, sendo x a sua parte real e y a parte imaginária, ambas representadas por valores reais. Crie um Tipo Abstrato de Dados (TAD) que represente os números complexos com as seguintes funções:
(a) criar um número complexo
(b) destruir um número complexo
(c) soma de dois números complexos;
(d) subtração de dois números complexos;
(e) multiplicação de dois números complexos;
(f) divisão de dois números complexos.