Pontifícia Universidade Católica
do Rio de Janeiro

INF 1007 – Programação II

Aula 12 – Tipos Abstratos de Dados

Edirlei Soares de Lima
<elima@inf.puc-rio.br>

visionlab
visualization laboratory

Tipo Abstrato de Dados (TAD)

- **Um TAD define:**
  - Um novo tipo de dado;
  - O conjunto de operações para manipular dados desse tipo;
- **Um TAD facilita:**
  - A manutenção e a reutilização de código;
  - A implementação das operações de TAD não precisa ser conhecida;
  - Para utilizar um TAD é necessário conhecer a sua funcionalidades, mas não a sua implementação;
- **Um TAD possui:**
  - Uma interface definindo o nome e as funcionalidades do TAD;
  - Uma implementação contendo a real implementação das funcionalidades do TAD;

Tipo Abstrato de Dados (TAD)

- **Exemplo: <u>TAD Ponto</u>** - Tipo de dado para representar um ponto no R^2 com as seguintes operações:
  - **cria** - cria um ponto com coordenadas x e y;
  - **libera** - libera a memória alocada por um ponto;
  - **acessa** - retorna as coordenadas de um ponto;
  - **atribui** - atribui novos valores às coordenadas de um ponto;
  - **distancia** - calcula a distância entre dois pontos;

Tipo Abstrato de Dados (TAD)

- **Interface do TAD Ponto:**
  - Define o nome do tipo e os nomes das funções exportadas;
  - A composição da estrutura Ponto não faz parte da interface:
    - Não é exportada pelo módulo;
    - Não faz parte da interface do módulo;
    - Não é visível para outros módulos;
  - Os módulos que utilizarem o TAD Ponto:
    - Não poderão acessar diretamente os campos da estrutura Ponto;
    - Só terão acesso aos dados obtidos através das funções exportadas.

Arquivo: ponto.h (interface do TAD Ponto)

```
/* TAD: Ponto (x,y) */
/* Tipo exportado */
typedef struct ponto Ponto;

/* Funções exportadas */
/* Função cria - Aloca e retorna um ponto com coordenadas (x,y) */
Ponto* pto_cria(float x, float y);

/* Função libera - Libera a memória de um ponto */
void pto_libera(Ponto* p);

/* Função acessa - Retorna as coordenadas de um ponto */
void pto_acessa(Ponto* p, float* x, float* y);

/* Função atribui - Atribui valores às coordenadas do ponto */
void pto_atribui(Ponto* p, float x, float y);

/* Função distancia - Retorna a distância entre dois pontos */
float pto_distancia(Ponto* p1, Ponto* p2);
```

Tipo Abstrato de Dados (TAD)

- **Implementação do TAD Ponto:**
  - Inclui o arquivo de interface de Ponto;
  - Define a composição da estrutura Ponto;
  - Inclui a implementação das funções externas;

Arquivo: ponto.c (implementação do TAD Ponto)

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "ponto.h"

struct ponto {
  float x;
  float y;
};

Ponto* pto_cria(float x, float y)
{
  Ponto* p = (Ponto*) malloc(sizeof(Ponto));
  if (p == NULL) {
    printf("Memória insuficiente!\n");
    exit(1);
  }
  p->x = x;
  p->y = y;
  return p;
}
```

```
void pto_libera(Ponto* p){
  free(p);
}

void pto_acessa(Ponto* p, float* x, float* y){
  *x = p->x;
  *y = p->y;
}

void pto_atribui(Ponto* p, float x, float y) {
  p->x = x;
  p->y = y;
}

float pto_distancia(Ponto* p1, Ponto* p2) {
  float dx = p2->x - p1->x;
  float dy = p2->y - p1->y;
  return sqrt(dx*dx + dy*dy);
}
```

Tipo Abstrato de Dados (TAD)

- **Exemplo de utilização do TAD Ponto:**

```
#include <stdio.h>
#include "ponto.h"

int main(void)
{
  float x, y;
  Ponto* p = pto_cria(2.0,1.0);
  Ponto* q = pto_cria(3.4,2.1);
  float d = pto_distancia(p,q);
  printf("Distancia entre pontos: %f\n",d);
  pto_libera(q);
  pto_libera(p);
  return 0;
}
```

Tipo Abstrato de Dados (TAD)

- **Exemplo: TAD Circulo** - Tipo de dado para representar um circulo com as seguintes operações:
  - **cria** - cria um círculo com centro (x,y) e raio r;
  - **libera** - libera a memória alocada por um círculo;
  - **area** - calcula a área do círculo;
  - **interior** - verifica se um dado ponto está dentro do círculo;

Arquivo: circulo.h (interface do TAD Circulo)

```
/* TAD: Círculo */
/* Dependência de módulos */
#include "ponto.h"

/* Tipo exportado */
typedef struct circulo Circulo;

/* Funções exportadas */
/* cria - Aloca e retorna um círculo com centro(x,y) e raio r */
Circulo* circ_cria(float x, float y, float r);

/* libera - Libera a memória de um círculo previamente criado */
void circ_libera(Circulo* c);

/* area - Retorna o valor da área do círculo */
float circ_area(Circulo* c);

/* interior - Verifica se um ponto p está dentro do círculo */
int circ_interior(Circulo* c, Ponto* p);
```

Arquivo: circulo.c (implementação do TAD Circulo)

```
#include <stdlib.h>
#include "circulo.h"

#define PI 3.14159

struct circulo {
  Ponto* p;
  float r;
};

Circulo* circ_cria(float x, float y, float r)
{
  Circulo* c = (Circulo*)malloc(sizeof(Circulo));
  c->p = pto_cria(x,y);
  c->r = r;
  return c;
}
```

```
void circ_libera(Circulo* c)
{
  pto_libera(c->p);
  free(c);
}

float circ_area(Circulo* c)
{
  return PI*c->r*c->r;
}

int circ_interior(Circulo* c, Ponto* p)
{
  float d = pto_distancia(c->p,p);
  return (d<c->r);
}
```

Tipo Abstrato de Dados (TAD)

- Podemos utilizar um TAD sem precisarmos conhecer a sua implementação.

- Exemplo: **TAD Matriz** - Tipo de dado para representar matrizes com as seguintes operações:
  - **cria**: operação que cria uma matriz de dimensão m por n;
  - **libera**: operação que libera a memória alocada para a matriz;
  - **acessa**: operação que acessa o elemento da linha i e da coluna j da matriz;
  - **atribui**: operação que atribui o elemento da linha i e da coluna j da matriz;
  - **linhas**: operação que devolve o número de linhas da matriz;
  - **colunas**: operação que devolve o número de colunas da matriz.

Arquivo: matriz.h (interface do TAD Matriz)

```
/* TAD: matriz m por n */
/* Tipo exportado */
typedef struct matriz Matriz;
/* Funções exportadas */
/* cria - Aloca e retorna uma matriz de dimensão m por n */
Matriz* mat_cria(int m, int n);

/* libera - Libera a memória de uma matriz previamente criada. */
void mat_libera(Matriz* mat);

/* acessa - Retorna o valor do elemento da linha i e coluna j */
float mat_acessa(Matriz* mat, int i, int j);

/* atribui - atribui o valor ao elemento da linha i e coluna j */
void mat_atribui(Matriz* mat, int i, int j, float v);

/* linhas - Retorna o número de linhas da matriz */
int mat_linhas(Matriz* mat);

/* colunas - Retorna o número de colunas da matriz */
int mat_colunas(Matriz* mat);
```

Tipo Abstrato de Dados (TAD)

- Verificar se uma matriz é **simétrica**: retorna 1 se verdadeiro e 0 se falso:
  - Uma matriz é **simétrica** se ela for igual a sua transposta;
  - Uma matriz **transposta** é o resultado da troca de linhas por colunas da matriz original;

```
int simetrica(Matriz *mat)
{
  int i, j;
  for (i=0; i<mat_linhas(mat); i++)
  {
    for (j=0; j<mat_colunas(mat); j++)
    {
      if (mat_acessa(mat,i,j) != mat_acessa(mat,j,i))
        return 0;
    }
  }
  return 1;
}
```

Tipo Abstrato de Dados (TAD)

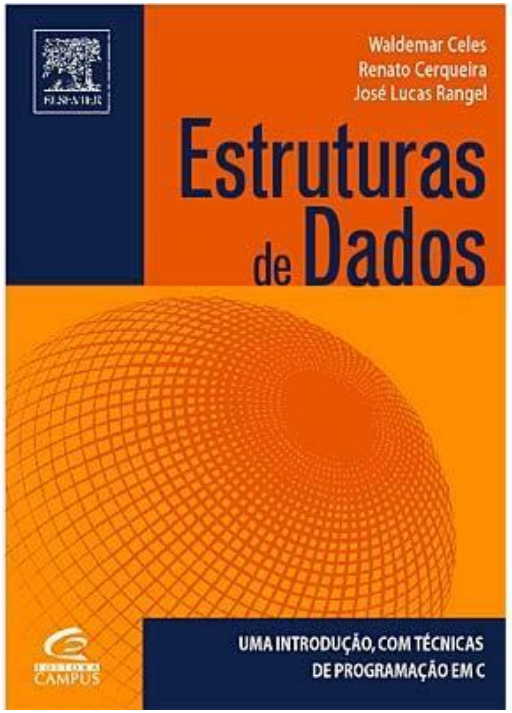
- **Multiplicar** uma matriz por um **escalar**:

```
Matriz *mult_matriz_escalar(Matriz *mat_a, float s)
{
  int i, j;
  Matriz *mat_b = mat_cria(mat_linhas(mat_a), mat_colunas(mat_a))
  for (i=0; i<mat_linhas(mat_a); i++)
  {
    for (j=0; j<mat_colunas(mat_a); j++)
    {
      mat_atribui(mat_b, i, j, s * mat_acessa(mat_a,i,j));
    }
  }
  return mat_b;
}
```

Leitura Complementar

- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, **Introdução a Estruturas de Dados**, Editora Campus (2004).

- **Capítulo 9 – Tipos Abstratos de Dados**



Exercícios

Lista de Exercícios 11 – Tipos Abstratos de Dados

http://www.inf.puc-rio.br/~elima/prog2/