

# Tipo Abstrato de Dados

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)  
Engenharia de Computação  
Departamento Acadêmico de Informática (Dainf)  
Universidade Tecnológica Federal do Paraná (UTFPR)  
Campus Pato Branco

- Estrutura de Dados
- Tipo Abstrato de Dados
- Modularização
  - Exemplo de TAD
- Exemplos de Erros Comuns em TAD

- Um programa em C pode ser dividido em vários arquivos fontes
- Programação modularizada
- Tipo de dados, estrutura de dados, tipo abstrato de dados (TAD)
  - Termos parecidos, mas com significados diferentes

## Estrutura de Dados

# Estrutura de Dados

- Tipo de dado: define um conjunto de valores que uma variável pode assumir
  - Por exemplo, uma variável do tipo `int` pode ter apenas valores inteiros entre -2.147.483.648 e +2.147.483.647
- Novos tipos de dados podem ser definidos em termos de outros já definidos
  - Tipos de dados estruturados, como vetores, matrizes, registros, etc

# Estrutura de Dados

- Estrutura de Dados: define a forma de organização e o relacionamento lógico entre tipos de dados
- Estruturas de dados podem ser:
  - Homogêneas: vetores, matrizes e strings
  - Heterogêneas: registros (*structs*) e uniões (*unions*)

- Exemplo de estrutura de dados: vetor (arranjo – *array*) de inteiros (int)
  - Cada elemento deve ser um número inteiro
  - Estrutura linear
  - Número finito de elementos
  - Acesso por índices
  - Para a remoção de elementos, podem haver deslocamento de elementos

- Em uma estrutura de dados, os campos/atributos juntos têm um significado




```
typedef struct {  
    char nome_disc[1001];  
    int RA;  
    float notas[3];  
}NotaDisciplina;
```



## Tipo Abstrato de Dados

# Tipo Abstrato de Dados

- Conjunto bem definido de estruturas de dados e do grupo de operações que podem ser aplicadas nesses dados

mun <u>do</u> real	dados de interesse	ESTRUTURA de armazenamento	possíveis OPERAÇÕES
 pessoa	<ul style="list-style-type: none"><li>a idade da pessoa</li></ul>	<ul style="list-style-type: none"><li>tipo inteiro</li></ul>	<ul style="list-style-type: none"><li>nasce ( <math>i \leftarrow - 0</math> )</li><li>aniversário ( <math>i \leftarrow i + 1</math> )</li></ul>
 cadastro de funcionários	<ul style="list-style-type: none"><li>o nome, cargo e o salário de cada funcionário</li></ul>	<ul style="list-style-type: none"><li>tipo lista ordenada</li></ul>	<ul style="list-style-type: none"><li>entra na lista</li><li>sai da lista</li><li>altera o cargo</li><li>altera o salário</li></ul>
 fila de espera	<ul style="list-style-type: none"><li>nome de cada pessoa e sua posição na fila</li></ul>	<ul style="list-style-type: none"><li>tipo fila</li></ul>	<ul style="list-style-type: none"><li>sai da fila (o primeiro)</li><li>entra na fila (no fim)</li></ul>

# Tipo Abstrato de Dados

- Estruturas de dados são implementadas para serem usados pelo programa por meio de operações apropriadas
- Muitas vezes é conveniente pensar nas operações suportadas por estruturas de dados em vez de na maneira como elas são implementadas
- Essas abstrações podem ser definidas em um tipo abstrato de dados (TAD)

# Tipo Abstrato de Dados

- Os dados armazenados devem ser manipulados apenas pelos operadores
  - Ocultamento dos detalhes de representação e implementação
  - Encapsulamento dos dados e do comportamento
  - Acesso somente às operações
  - Reutilização e flexibilidade

# Tipo Abstrato de Dados

- Exemplos de operações:
  - Criação de estrutura
  - Inclusão de um elemento
  - Remoção de um elemento
  - Acesso a um elemento
  - Etc

# Tipo Abstrato de Dados

- Exemplo de TAD disponível na biblioteca *stdio.h*
  - Arquivos em C: *FILE \*arq*;
  - Acesso aos dados de *arq* somente por funções de manipulação do tipo *FILE*
    - *fopen()*
    - *fclose()*
    - *fscanf()*
    - etc

## Modularização

- Por convenção, os TADs são construídos em arquivos separados
- São utilizados arquivos de cabeçalho (.h) e de código fonte (.c) para implementação de TADs
- Por convenção, o arquivo de cabeçalho e de código fonte devem ter o mesmo nome, alterando apenas a extensão
  - `minha_implementacao.h`
  - `minha_implementacao.c`



# Modularização

- Toda a manipulação da estrutura de dados deve ser feita por meio de funções que interagem com ela
  - Possibilita "esconder" a implementação
  - Quem usa o TAD, precisa apenas conhecer as funcionalidades que ele implementa
  - Facilita manutenção e reutilização

- Exemplo de código a ser melhorado

```
typedef struct {
    char nome_disc[1001];
    int RA;
    float notas[3];
}NotaDisciplina;

int main() {
    NotaDisciplina nd;
    char nome_disc[101];
    int RA;
    float notas[3];

    printf("Nome disciplina: ");
    scanf("%s", nome_disc);
    printf("Numero matricula: ");
    scanf("%d", &RA);
    printf("Informe 3 notas do aluno: ");
    scanf("%f %f %f", &notas[0], &notas[1], &notas[2]);
    strcpy(nd.nome_disc, nome_disc);
    nd.RA = RA;
    nd.notas[0] = notas[0];
    nd.notas[1] = notas[1];
    nd.notas[2] = notas[2];
}
```

- Código com modularização

```
typedef struct {
    char nome_disc[100];
    int RA;
    float notas[3];
}NotaDisciplina;

NotaDisciplina cadastrar(char nome[], int RA, float notas[]){
    NotaDisciplina disc;

    strcpy(nd .nome_disc, nome_disc);
    nd.RA = RA;
    nd.notas[0] = notas[0];
    nd.notas[1] = notas[1];
    nd.notas[2] = notas[2];

    return disc;
}

int main() {
    NotaDisciplina nd;
    char nome_disc[101];
    int RA;
    float notas[3];

    ...

    nd = cadastrar(nome, RA, notas);
}
```

- Arquivo .h:
  - Protótipos das funções
  - *Typedefs*
  - Variáveis globais
- Arquivo .c:
  - Declaração dos tipos de dados (ou isso pode ser feito no arquivo .h)
  - Implementação das funções

# Modularização

## Exemplo de TAD

- Criar um TAD para trabalhar com pontos
  - Definir tipo de dado
  - Definir operações que serão utilizadas

- Criar um TAD para trabalhar com pontos
  - Definir tipo de dado
    - Estrutura (struct) que armazene coordenadas X e Y
  - Definir operações que serão utilizadas
    - Criar um ponto no espaço da memória
    - Atribuir valores ao ponto (adicionar valores de X e Y)
    - Alterar valores do ponto (receber valores de X e Y)
    - Acessar valores do ponto
    - Calcular distância entre pontos (distância euclidiana)

- Primeiro passo: definir arquivo .h
  - *Typedefs*
  - Protótipos das funções
  - Variáveis globais

# Modularização

## Exemplo de TAD

- Primeiro passo: definir arquivo .h
  - Tipos de ponteiro
  - Protótipos das funções
  - Variáveis globais
- Tipo de dado:

```
typedef struct {  
    float x, y;  
}Ponto;
```



# Modularização

## Exemplo de TAD

- Primeiro passo: definir arquivo .h

```
// arquivo Ponto.h
typedef struct {
    float x, y;
}Ponto;

Ponto* criar(float x, float y);

int acessar(Ponto *p, float *x, float *y);

int alterar(Ponto *p, float x, float y);

float ponto_x(Ponto *p);

float ponto_y(Ponto *p);

void imprimir_ponto(Ponto *p);

float distancia(Ponto *p1, Ponto *p2);
```

- Por que usar ponteiros?

# Modularização

## Exemplo de TAD

- Segundo passo: definir arquivo .c
  - Implementação das funções

```
// Ponto.c
#include <math.h>
#include "Ponto.h"

Ponto* criar(float x, float y){
    Ponto *p = (Ponto*) malloc(sizeof(Ponto));

    p->x = x;
    p->y = y;

    return p;
}
```

# Modularização

## Exemplo de TAD

```
int acessar(Ponto *p, float *x, float *y){  
    if (p == NULL)  
        return 0;  
  
    *x = p->x;  
    *y = p->y;  
  
    return 1;  
}
```

```
int alterar(Ponto *p, float x, float y){  
    if (p == NULL)  
        return 0;  
  
    p->x = x;  
    p->y = y;  
  
    return 1;  
}
```

# Modularização

## Exemplo de TAD

```
float ponto_x(Ponto *p) {  
    if (p != NULL)  
        return p->x;  
    return 0;  
}  
  
float ponto_y(Ponto *p) {  
    if (p != NULL)  
        return p->y;  
    return 0;  
}  
  
void imprimir_ponto(Ponto *p) {  
    if (p != NULL)  
        printf("X: %f Y: %f", p->x, p->y);  
}
```

# Modularização

## Exemplo de TAD

```
float distancia(Ponto *p1, Ponto *p2){  
    if ((p1 == NULL) || (p2 == NULL))  
        return -1;  
  
    float dx = p1->x - p2->x;  
    float dy = p1->y - p2->y;  
  
    return sqrt(dx * dx + dy * dy);  
}
```

# Modularização

## Exemplo de TAD

```
// main.c
#include <math.h>
#include "Ponto.h"

int main(void){
    float d;
    Ponto *p1 = criar(15, 25);
    Ponto *p2 = criar(18, 9);

    alterar(p1, 10, 20);

    d = distancia(p1, p2);

    printf("distancia entre pontos: %d\n", d);

    return 0;
}
```

- Repare que aqui são utilizados ponteiros, já que nos códigos que utilizam "*Ponto.h*", o tipo Ponto não é reconhecido, ou seja, não há como saber o espaço necessário as estruturas
  - Encapsulamento

# Exemplos de Erros Comuns em TAD

```
typedef struct {  
    float x, y;  
}Ponto;  
Ponto criar(float x, float y);  
int acessar(Ponto p, float *x, float *y);  
int alterar(Ponto p, float x, float y);  
float distancia(Ponto p1, Ponto p2);
```

- Usar novos tipos de dados como parâmetros e retorno de funções em TADs
- Usar ponteiros

```
typedef struct {  
    float x, y;  
}Ponto;  
Ponto* criar(float x, float y);  
int acessar(Ponto *p, float *x, float *y);  
int alterar(Ponto *p, float x, float y);  
float distancia(Ponto *p1, Ponto *p2);
```

# Exemplos de Erros Comuns em TAD

```
#include <math.h>
#include "Ponto.h"

int main(void){
    Ponto p1 = criar(15, 25);
    return 0;
}
```

- Caso a função "*criar*" seja chamada em um outro arquivo fonte que não seja o seu respectivo TAD, ocorrerá erro, já que o tipo "*Ponto*" não é conhecido dentro do escopo do arquivo
- Solução: usar ponteiro

```
#include <math.h>
#include "Ponto.h"

int main(void){
    Ponto *p1 = criar(15, 25);
    return 0;
}
```



# Exemplos de Erros Comuns em TAD

```
#include <math.h>
#include "Ponto.h"

int main(void){
    Ponto *p1 = criar(15, 25);
    p1->x = 10;
    return 0;
}
```

- Tentar acessar a um campo específico de uma *struct* fora do seu respectivo TAD
- Solução: criar uma função que receba ponteiro referente à estrutura que retorne ou altere o campo desejado

```
#include <math.h>
#include "Ponto.h"

int main(void){
    Ponto *p1 = criar(15, 25);
    alterar_x(p1, 10);
    return 0;
}
```

# Exercício

- Altere o TAD de pontos para trabalhar em um espaço 3D
  - Variáveis:  $x$ ,  $y$ ,  $z$
  - Adapte as funções pra trabalhar com as novas variáveis

- Crie um Tipo Abstrato de Dados (TAD) que represente os números racionais e que contenha as seguintes funções:
  - Criar racional
  - Soma racionais (não altere os números originais)
  - Multiplica racionais (não altere os números originais)
  - Teste se dois números racionais são iguais

- Implemente um tipo abstrato de dados para vetores de números inteiros contendo as seguintes funções:
  - Criar um vetor de  $n$  números
  - Retorna o maior número de um vetor
  - Retorna o menor número de um vetor
  - Retorna a média de um vetor
  - Retorna a soma de dois vetores
  - Retorna o produto interno de dois vetores



Prata, S.

*C++ Primer Plus.*

Waite Group Press, 1998.



Szwarcfiter, J.; Markenzon, L.

*Estruturas de Dados e Seus Algoritmos.*

LTC, 2010.



Tenenbaum, A.; Langsam, Y.

*Estruturas de Dados usando C.*

Pearson, 1995.