



**UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI**  
**CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB**  
**CURSO DE SISTEMAS DE INFORMAÇÃO**  
**PICOS - PI**

## **TIPOS ABSTRATOS DE DADOS (TAD)**

Prof. Ma. Luana Batista da Cruz  
luana.b.cruz@nca.ufma.br

# Roteiro

2

- Módulos e compilação em separado
- O que é TAD?
- Vantagens de usar
- Como criar?

# Módulos e compilação em separado

3

## ❑ **Módulo**

- Um arquivo com funções que representam apenas parte da implementação de um programa completo

## ❑ **Arquivo objeto**

- Resultado de compilar um módulo
- Geralmente com extensão .o ou .obj

## ❑ **Ligador**

- Junta todos os arquivos objeto em um único arquivo executável

# Módulos e compilação em separado

4

- Um módulo na linguagem C pode ser criado utilizando arquivos de cabeçalho e de implementação
  - **Arquivo de cabeçalho:** os arquivos '.h' são utilizados para especificar assinatura de funções, definições de constantes, tipos de dados criados pelo usuário etc. De modo geral, os arquivos de cabeçalho tem como função definir a interface de um módulo
  - **Arquivo de implementação:** Os arquivos '.c' implementam as funções definidas na interface. De modo geral, esses arquivos são compostos por diversas funções e estruturas de dados utilizadas internamente

# Módulos e compilação em separado

5

- ❑ Interface de um módulo de funções:
  - ❑ Arquivo contendo apenas:
    - Os protótipos das funções oferecidas pelo módulo
    - Os tipos de dados exportados pelo módulo (typedef's, struct's, etc)
- ❑ Em geral possui:
  - nome: igual ao do módulo ao qual está associado
  - extensão: .h

# Módulos e compilação em separado

## Exemplo

6

### ❏ **operacoes.h**

- Assinatura das funções: “soma”, “subtração”, “multiplicação” e “divisão”

```
int soma(int a, int b);  
int subtracao(int a, int b);  
int multiplicacao(int a, int b);  
int divisao(int a, int b);
```

# Módulos e compilação em separado

## Exemplo

7

### ❏ **operacoes.c:**

- Arquivo com a implementação das funções de manipulação de inteiros: “soma”, “subtração”, “multiplicação” e “divisão”

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

int soma(int a, int b){
    return a+b;
}
int subtracao(int a, int b){
    return a-b;
}
int multiplicacao(int a, int b){
    return a*b;
}
int divisao(int a, int b){
    return a/b;
}
```

# Módulos e compilação em separado

## Exemplo

8

### ❏ **main.c**

- O main.c precisa conhecer os protótipos declarados em .h

```
#include <stdlib.h>
#include <stdio.h>
#include "operacoes.h"

int main(){
    int so = soma(2,3);
    int su = subtracao(2,3);
    int mu = multiplicacao(2,3);
    int di = divisao(2,3);

    printf("A soma eh: %d\n", so);
    printf("A multiplicacao eh: %d\n", mu);
    printf("A subtracao eh: %d\n", su);
    printf("A divisao eh: %d\n", di);
    return 0;
}
```




# TAD

9

- ❑ Um TAD define:
  - Um tipo de dados
  - O conjunto de operações para manipular dados desse tipo
  
- ❑ Um TAD pode ser definido por uma coleção bem definida de dados a serem armazenados, e um grupo de operadores que podem ser aplicados para manipulação desses dados

- ❑ Da mesma forma, pode ser visto como um modelo matemático que encapsula um modelo de dados e um conjunto de procedimentos que atuam com exclusividade sobre os dados encapsulados
  
- ❑ **Características fundamentais:**
  - Os operadores do TAD implementam regras bem definidas para manipulação dos valores armazenados
  - Os valores armazenados devem ser manipulados EXCLUSIVAMENTE pelos operadores do TAD

## ❏ Exemplo (analogia à TV)

Mundo real	Coleção bem definida de dados	Grupo de operadores
	<ul style="list-style-type: none"><li>• Som, canal</li></ul>	<ul style="list-style-type: none"><li>• Aumentar volume</li><li>• Diminuir volume</li><li>• Mudar de canal (canal acima)</li><li>• Mudar de canal (canal abaixo)</li></ul>

# Vantagens do TAD

12

- ❑ **Abstração da informação:** permite uma melhor compreensão dos algoritmos e maior facilidade de programação
- ❑ **Mais seguro programar:** apenas as operações do TAD alteram os dados
- ❑ **Reúso:** uma vez definido, implementado e testado, o TAD pode ser acessado por diferentes programas
- ❑ **Manutenção:** mudanças na implementação do TAD não afetam o código fonte dos programas que o utilizam

# Práticas de programação de TAD

13

- ❑ Para implementar um TAD em C, usa-se a definição de tipos juntamente com a implementação de funções que agem sobre aquele tipo
- ❑ Como boa prática de programação, evita-se acessar o dado diretamente, fazendo o acesso somente através de das funções
  - O TAD encapsula a estrutura de dados. Os usuários do TAD só tem acesso a algumas operações disponibilizadas sobre esses dados

## ❑ A interface de um TAD define:

- O nome do tipo
- Os nomes das funções exportadas
- Os nomes das funções devem ser prefixada pelo nome do tipo, evitando conflitos quando tipos distintos são usados em conjunto
  - **pto\_cria** - função para criar um tipo Ponto
  - **circ\_cria** - função para criar um tipo Circulo

## ❏ Implementação de um TAD:

- ❑ O arquivo de implementação de um TAD deve incluir o arquivo de interface do TAD:
  - Permite utilizar as definições da interface, que são necessárias na implementação
  - Garante que as funções implementadas correspondem às funções da interface
  - Compilador verifica se os parâmetros das funções implementadas equivalem aos parâmetros dos protótipos
- ❑ E deve também incluir as variáveis globais e funções auxiliares

# TAD - Ponto

16

- ❑ Tipo de dado para representar um ponto 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



# TAD - Interface Ponto

17

- ❑ 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

# TAD - Interface Ponto

18

## □ Ponto.h (arquivo com a interface de 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 previamente criado */  
void pto_libera(Ponto* p);  
  
/* Função acessa - Retorna os valores das coordenadas de um ponto */  
void pto_acessa(Ponto* p, float x, float y);  
  
/* Função atribui - Atribui novos valores às coordenadas de um 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);
```

# TAD - Implementação do Ponto

19

- ❑ Inclui o arquivo de interface de Ponto
- ❑ Define a composição da estrutura Ponto
- ❑ Inclui a implementação das funções externas

# TAD - Implementação do Ponto

20

## ❑ Ponto.c (arquivo com o TAD ponto)

```
#include <stdlib.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("Memoria insuficiente!\n");
        exit(1);
    }
    p->x = x;
    p->y = y;
    return p;
}
```

# TAD - Implementação do Ponto

21

## □ Ponto.c (arquivo com o TAD ponto) Continuação..

```
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);
}
```

# TAD - Implementação do Ponto

22

## □ main.c

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

int main(){
    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;
}
```

# Atividade

23

- ❑ Implemente um TAD ContaBancaria, com os campos **número** e **saldo** onde os clientes podem fazer as seguintes operações:
  - Iniciar uma conta com um número e saldo inicial
  - Depositar um valor
  - Sacar um valor
  - Imprimir o saldo
  
- ❑ Faça um pequeno programa para testar o seu TAD

# Atividade

24

## ❏ **contaBancaria.h**

```
typedef struct ContaBancaria ContaBancaria;  
  
// cabeçalho das funções  
void Inicializa(ContaBancaria* pconta, int numero, double saldo);  
void Deposito(ContaBancaria* pconta, double valor);  
void Saque(ContaBancaria* pconta, double valor);  
void Imprime(ContaBancaria conta);
```



# Atividade

25

## ❏ **contaBancaria.c**

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

struct ContaBancaria{
    int numero;
    double saldo;
};

void Inicializa(ContaBancaria* pconta, int numero, double saldo){
    pconta->numero = numero;
    pconta->saldo = saldo;
}

void Deposito(ContaBancaria* pconta, double valor){
    pconta->saldo += valor;
}
```

# Atividade

26

## ❏ **contaBancaria.c (Continuação..)**

```
void Saque(ContaBancaria* pconta, double valor){
    pconta->saldo -= valor;
}
void Imprime(ContaBancaria conta){
    printf("Numero: %d\n", conta.numero);
    printf("Saldo: %f\n", conta.saldo);
}
```

# Atividade

27

## ❏ main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "contaBancaria.h"

int main (){
    ContaBancaria conta1;
    Inicializa(&conta1, 918556, 300.00);
    printf("\nAntes da movimentacao:\n ");
    Imprime(conta1);
    Deposito(&conta1, 50.00);
    Saque(&conta1, 70.00);
    printf("\nDepois da movimentacao:\n ");
    Imprime (conta1);
    return 0;
}
```

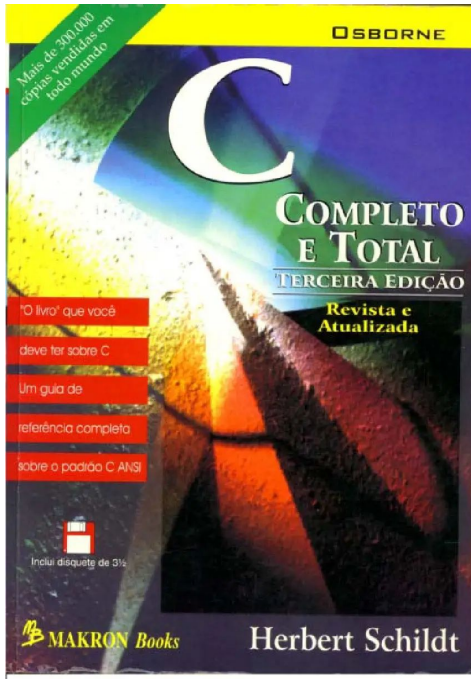
# Atividade

28

- ❑ Modifique o TAD de modo que as seguintes operações tenham validações:
  - Inicializa: não pode inicializar uma conta com saldo negativo
  - Depositar: não pode depositar com saldo negativo
  - Sacar: não pode sacar se saldo for insuficiente
  
- ❑ Faça um pequeno programa para testar o seu TAD modificado

# Referências

29



SCHILD, Herbert. **C completo e total**. Makron, 3ª edição revista e atualizada, 1997.



SZWARCHFITER, J. **Estruturas de Dados e seus algoritmos**. 3 ed. Rio de Janeiro: LTC, 2010.