

# TIPO ABSTRATO DE DADO (TAD)

Prof. Muriel Mazzetto  
Estrutura de Dados

# Estruturas de Dados

2

- Tipo de dado: define um conjunto de valores (domínio) que uma variável pode assumir.
- Estrutura de Dados: define um relacionamento lógico entre tipos de dados.
- Estrutura de Dados: é um modo particular de armazenamento e organização de dados em um computador de modo que possam ser usados eficientemente, facilitando sua busca e modificação.  
(Wikipédia)

# Estruturas de Dados

3

- Homogêneos: conjuntos formados pelos mesmos tipos de dados:
  - ▣ Vetores, Matrizes e Strings;
  
- Heterogêneos: conjuntos formados por diferentes tipos de dados:
  - ▣ Registros (Structs)

# Estruturas de Dados

4

- Vetores (arrays):
  - ▣ Estruturas lineares e estáticas;
  - ▣ Número finito de elementos;
  - ▣ Acesso por índices (rápido);
  - ▣ Remoção lenta.
  - ▣ Recomendados para pouca alteração dos dados.

# Tipo Abstrato de Dado




5

- Conjunto bem definido das estruturas e do grupo de operações que podem ser aplicados nelas.

# Tipo Abstrato de Dado

6

- Conjunto bem definido das estruturas e do grupo de operações que podem ser aplicados nelas.

mundo 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 Dado

7

- Os dados armazenados podem ser manipulados apenas pelos operadores.
- Ocultamento dos detalhes de representação e implementação, apenas funcionalidade é conhecida.
- Encapsulamento dos dados e do comportamento.
- Acesso somente às operações, e não diretamente aos dados.
- Reutilização e flexibilidade do TAD em diferentes aplicações.

# Tipo Abstrato de Dado

8

- Operações mais comuns utilizadas:
  - ▣ Criação de estrutura;
  - ▣ Inclusão de um elemento;
  - ▣ Remoção de um elemento;
  - ▣ Acesso a um elemento;



# Exemplo de TAD

9

- Arquivos em C: **FILE** \* **arq;**
- Acesso aos dados de *arq* somente por funções de manipulação do tipo *FILE*.
  - ▣ fopen();
  - ▣ fclose();
  - ▣ fscanf();
  - ▣ etc.

# Modularização

10

- Por convenção os TADs são construídos em arquivos separados.
- Utilizam-se arquivos de **cabeçalho (.h)** e de **código fonte (.c)** para modularizar.
- **LEMBRE-SE:** o arquivo de cabeçalho e de código fonte devem ter o mesmo nome, alterando apenas a extensão.
  - Biblioteca\_nova.h
  - Biblioteca\_nova.c

# Modularização

11

- Possibilita “esconder” a implementação.
- Quem usa o TAD, precisa apenas conhecer as funcionalidades que ele implementa.
- Facilita manutenção e reutilização.

# Modularização

12

- Arquivo .h:
  - ▣ protótipos das funções.
  - ▣ tipos de ponteiro.
  - ▣ variáveis globais.
  
- Arquivo .c:
  - ▣ declaração dos tipos de dados.
  - ▣ implementação das funções.

# Exemplo de TAD

13

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

# Exemplo de TAD

14

- 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 (alocar memória).
    - Deletar um ponto do espaço (liberar memória).
    - Atribuir valores ao ponto (adicionar valores de X e Y).
    - Acessar valores do ponto (receber valores de X e Y).
    - Calcular distância entre pontos (distância euclidiana).

# Exemplo de TAD

15

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

# Exemplo de TAD: Ponto.h

16

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

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



# Exemplo de TAD: Ponto.h

17

- Primeiro passo: definir arquivo .h

```
//Arquivo Ponto.h
typedef struct ponto Ponto;

//Cria um novo ponto
Ponto* criar_ponto(float x, float y);
//Libera um ponto
void deletar_ponto(Ponto* p);
//Acessa os valores "x" e "y" de um ponto
int acessar_ponto(Ponto* p, float* x, float* y);
//Atribui os valores "x" e "y" a um ponto
int atribuir_ponto(Ponto* p, float x, float y);
//Calcula a distância entre dois pontos
float distancia(Ponto* p1, Ponto* p2);
```

# Exemplo de TAD: Ponto.c

18

- Segundo passo: definir arquivo .c
  - ▣ Declaração dos tipos de dados.
  - ▣ Implementação das funções.
- Tipo de dado a ser utilizado:

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

# Exemplo de TAD: Ponto.c

19

```
//Arquivo .c
#include <stdlib.h>
#include <math.h>
#include "Ponto.h" //inclui os Protótipos

//Definição do tipo de dados
struct ponto{
    float x;
    float y;
};
```

# Exemplo de TAD: Ponto.c

20

```
//Aloca e retorna um ponto com coordenadas "x" e "y"
Ponto* criar_ponto(float x, float y){
    Ponto* p = (Ponto*) malloc(sizeof(Ponto));
    if(p != NULL){
        p->x = x;
        p->y = y;
    }
    return p;
}

//Libera a memória alocada para um ponto
void deletar_ponto(Ponto* p){
    free(p);
}
```

# Exemplo de TAD: Ponto.c

21

```
//Atribui a um ponto as coordenadas "x" e "y"  
int atribuir_ponto(Ponto* p, float x, float y) {  
    if(p == NULL)  
        return 0;  
    p->x = x;  
    p->y = y;  
    return 1;  
}
```

# Exemplo de TAD: Ponto.c

22

```
//Recupera, por referência, o valor de um ponto
int acessar_ponto(Ponto* p, float* x, float* y){
    if(p == NULL)
        return 0;
    *x = p->x;
    *y = p->y;
    return 1;
}
```

# Exemplo de TAD: Ponto.c

23

```
//Calcula a distância entre dois pontos
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);
}
```

# Exemplo de TAD: main.c

24

```
#include <stdio.h>
#include <stdlib.h>
#include "Ponto.h"
int main(void) {
    float d;
    Ponto *p, *q;
    //Ponto r; //ERRO: NÃO ACEITA DECLARAR VARIÁVEL
    p = criar_ponto(10, 5);
    q = criar_ponto(5, 10);
    //q->x = 2; //ERRO: NÃO ACEITA ATRIBUIÇÃO DIRETA
    d = distancia(p, q);
    printf("Distancia entre pontos: %f\n", d);

    deletar_ponto(q);
    deletar_ponto(p);

    system("pause");
    return 0;
}
```



# Exercícios

25

- Altere a TAD de pontos para trabalhar em um espaço 3D.
  - ▣ Observe que serão variáveis  $x$ ,  $y$  e  $z$ .
  - ▣ Adeque as funções pra trabalhar com as novas variáveis.

# Exercícios

26

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