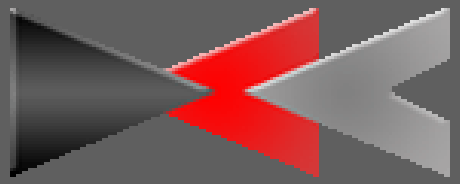
DCC107



Laboratório de Programação II

Tipos Abstratos de Dados (TAD) em C

Nesta Aula

- TAD – Tipos Abstratos de Dados;
- Exemplos (implementações de TAD):
  - TAD Ponto;
  - TAD Vetor com limites quaisquer em C;
  - TAD Numero Racional (exercício site monitoria).

TAD – Tipos Abstratos de Dados

- A idéia de TAD é desvincular o tipo de dado de sua implementação;
- Algumas vantagens desta desvinculação são:
  - Integridade dos dados;
  - Invisibilidade (representação escondida e inacessível);
  - Proteção (o cliente manipula os objetos através das operações fornecidas);
  - Facilidade de manutenção;
  - Reutilização.
- Um TAD está desvinculado de sua implementação, ou seja, quando definimos um TAD estamos preocupados com **o que** ele faz e não **como** ele faz.

Exemplo 1: TAD Ponto

tipo TADPonto

    domínio: REAL x REAL;

    operações:

        cria(x, y) → TADPonto;

        libera(TADPonto);

        acessa(TADPonto) → x, y;

        atribui(x, y) → TADPonto;

        distancia(TADPonto, TADPonto) → REAL;

    fim-operações;

fim-tipo.

Exemplo 1: TAD Ponto

- Construir dois arquivos para implementar o TADPonto:
  - **`ponto.h`** (interface do TAD);
  - **`ponto.c`** (implementação das operações).
- Isso é necessário para ser ter (como já dito anteriormente):
  - Invisibilidade; e
  - Proteção.

Exemplo 1: TAD Ponto

- Arquivo **`ponto.h`**:
  - Contém a definição do tipo Ponto (TAD ponto) e os protótipos das operações (o que o cliente pode usar):

```
#ifndef PONTO_H_INCLUDED
#define PONTO_H_INCLUDED

//TAD: Ponto (x, y)
typedef struct ponto Ponto;

//aloca e retorna um ponto com coordenadas (x, y)
Ponto* cria (float x, float y);
```

Exemplo 1: TAD Ponto

- Arquivo **`ponto.h`** (continuação):

```
//libera a memoria de um ponto previamente criado
void libera (Ponto* p);

//devolve os valores das coordenadas de um ponto
void acessa (Ponto* p, float* x, float* y);

//atribui novos valores as coordenadas de um ponto
void atribui (Ponto* p, float x, float y);

//retorna a distancia entre dois pontos
float distancia (Ponto* p1, Ponto* p2);

#endif //PONTO_H_INCLUDED
```

Exemplo 1: TAD Ponto

- Arquivo **`ponto.c`**. Contém a `struct` ponto (domínio) e as implementação das operações:

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

//definir a estrutura ponto da seguinte forma
struct ponto
{
    float x;
    float y;
};
```

Exemplo 1: TAD Ponto

- Arquivo **`ponto.c`** (continuação):

```
//alocar a estrutura do ponto e inicializar os seus campos
Ponto* 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;
}
```

Exemplo 1: TAD Ponto

- Arquivo **`ponto.c`** (continuação):

```
//libera a estrutura que foi criada atraves da funcao cria
void libera (Ponto* p)
{
    free(p);
}

//acessa e atribui valores as coordenadas de um ponto
void acessa (Ponto* p, float* x, float* y)
{
    *x = p->x;
    *y = p->y;
}
```

Exemplo 1: TAD Ponto

- Arquivo **`ponto.c`** (continuação):

```
void atribui (Ponto* p, float x, float y)
{
    if (p == NULL) //cria o ponto p
        p = cria(x, y);
    else
    {
        p->x = x;
        p->y = y;
    }
}
```

Exemplo 1: TAD Ponto

- Arquivo **`ponto.c`** (continuação):

```
//calcula a distancia entre os pontos p1 e p2
float distancia (Ponto* p1, Ponto* p2)
{
    float dx, dy;
    dx = p2->x - p1->x;
    dy = p2->y - p1->y;
    return sqrt(dx * dx + dy * dy);
}
```

Exemplo 1: TAD Ponto

- Arquivo **`main.c`**. Arquivo "cliente" do TAD Ponto, ou seja, local onde será usado:

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

int main()
{
    Ponto *p1, *p2, *p3 = NULL;
    float dist, x, y;

    //le as coordenadas de p1 e cria-o
    printf("\nDigite as coordenadas do ponto p1 (x y): ");
    scanf("%f %f", &x, &y);
    p1 = cria(x, y);
```

Neste arquivo, ou em qualquer cliente do TAD ponto, não se tem acesso aos campos `x` e `y` do TAD ponto (**invisibilidade** e **proteção**)

Exemplo 1: TAD Ponto

- Arquivo **`main.c`** (continuação):

```
//le as coordenadas de p2 e cria-o
printf("\nDigite as coordenadas do ponto p2 (x y): ");
scanf("%f %f", &x, &y);
p2 = cria(x, y);

atribui(p3, x, y);

acessa(p1, &x, &y);
printf("\nCoordenada x de p1: %.2f", x);
printf("\nCoordenada y de p1: %.2f\n", y);

acessa(p2, &x, &y);
printf("\nCoordenada x de p2: %.2f", x);
printf("\nCoordenada y de p2: %.2f\n", y);
```

Exemplo 1: TAD Ponto

- Arquivo **`main.c`** (continuação):

```
    //calcula e imprime a distancia entre p1 e p2
    dist = distancia(p1, p2);
    printf("\nDistancia de p1 a p2: %.2f\n", dist);

    //libera a memoria ocupada por p1, p2 e p3, nunca esquecer!
    libera(p1);
    libera(p2);
    libera(p3);
    return 0;
}
```

Exemplo 2: TAD VetorLim

- Em C, ao declarar um vetor `int vet[N]`, os índices deste vetor variam de `0` até `N - 1`. Além disso, estes índices não são testados para saber se encontram no intervalo `0` a `N - 1`.
- Desenvolver um **TAD VetorLim** em C de forma que os índices deste vetor variem de `li` (limite superior) a `ls` (limite superior) e que, ao acessar um elemento, o índice seja verificado se está entre `li` e `ls`.

Exemplo 2: TAD VetorLim

tipo VETOR

    domínio: VETOR, ÍNDICE, VALOR;

    operações:

        cria_vetor → VETOR;

        libera_vetor (VETOR);

        consulta(VETOR, ÍNDICE) → VALOR;

        atribui(VETOR, ÍNDICE, VALOR) → VETOR;

    fim-operações;

fim-tipo;

Exemplo 2: TAD VetorLim

- Construir dois arquivos para implementar o TAD VetorLim:
  - **`VetorLim.h`** (interface do TAD);
  - **`VetorLim.c`** (implementação das operações).

Exemplo 2: TAD VetorLim

- Arquivo **`VetorLim.h`**. Contém a definição do tipo e das operações:

```
typedef struct VetorLim TVetorLim;

//cria o tipo de dado TVetorLim dados os limites inferior e superior
TVetorLim* cria_vetor(int val_min, int val_max);

//libera a area de memória usada por TVetorLim
void libera_vetor(TVetorLim* vetor);

//consulta o elemento cujo indice eh "indice"
int consulta (TVetorLim* vetor, int indice);

//atribui na posicao indice o valor "valor"
void atribui (TVetorLim* vetor, int indice, int valor);
```

Exemplo 2: TAD VetorLim

- Arquivo **VetorLim.c**. Contém a struct VetorLim (domínio) e as implementação das operações:

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

struct VetorLim
{
   int lim_inf, lim_sup; //limites inferior e superior do vetor
   int *itens; //vetor que contem os valores a serem armazenados
};
```

Exemplo 2: TAD VetorLim

- Arquivo **`VetorLim.c`** (continuação):

```
TVetorLim* cria_vetor(int val_min, int val_max)
//cria o tipo de dado TVetorLim dados os limites inferior e superior
{
    int tam; //numero de elementos do vetor itens
    TVetorLim* vetor;

    //aloca memoria para a struct VetorLim
    vetor = (TVetorLim*) malloc(sizeof(TVetorLim));

    //determina o numero de elementos do vetor
    if(val_min < val_max)
    {
        vetor->lim_inf = val_min;
        vetor->lim_sup = val_max;
```

Exemplo 2: TAD VetorLim

- Arquivo **`VetorLim.c`** (continuação):

```
        //determina o numero de elementos do vetor
        tam = val_max - val_min + 1;
        //aloca memoria para tam int em itens
        vetor->itens = (int*) malloc(tam * sizeof(int));
        return vetor;
    }
    else
    {
        printf("Limite inferior %d >= limite superior %d",
            val_min, val_max);
        return NULL;
    }
}
```

Exemplo 2: TAD VetorLim

- Arquivo **`VetorLim.c`** (continuação):

```
void libera_vetor(TVetorLim* vetor)
//libera a area de memoria usada por TVetorLim
{
    free(vetor->itens);
    free(vetor);
}
```

Exemplo 2: TAD VetorLim

- Arquivo **`VetorLim.c`** (continuação):

```
int consulta (TVetorLim* vetor, int indice)
//consulta o elemento cujo indice eh "indice"
{
    int k;
    //verifica se o indice eh valido
    if(indice >= vetor->lim_inf && indice <= vetor->lim_sup)
    {
        //k eh o indice entre 0 e tam - 1
        k = indice - vetor->lim_inf;
        return vetor->itens[k];
    }
```

Exemplo 2: TAD VetorLim

- Arquivo **`VetorLim.c`** (continuação):

```
    else
    {
        printf("Indice: %d fora dos limites (%d, %d)", indice,
            vetor->lim_inf, vetor->lim_sup);
        exit(0);
    }
}
```

Exemplo 2: TAD VetorLim

- Arquivo **`VetorLim.c`** (continuação):

```
void atribui (TVetorLim* vetor, int indice, int valor)
//atribui na posicao indice o valor "valor"
{
    int k;
    //verifica se o indice eh valido
    if(indice >= vetor->lim_inf && indice <= vetor->lim_sup)
    {
        //k eh o indice entre 0 e tam - 1
        k = indice - vetor->lim_inf;
        vetor->itens[k] = valor;
    }
```

Exemplo 2: TAD VetorLim

- Arquivo **`VetorLim.c`** (continuação):

```
    else
    {
        printf("Indice: %d fora dos limites (%d, %d)", indice,
                vetor->lim_inf, vetor->lim_sup);
        exit(0);
    }
}
```

Exemplo 2: TAD VetorLim

- Arquivo **`main.c`**. "Cliente" do TAD VetoLim:

```c
#include <stdio.h>
#include <stdlib.h>
#include "VetorLim.h"
int main()
{
    int i;
    TVetorLim* vetor;
    vetor = cria_vetor(-29, 30);
    for(i = -29; i <= 30; i++)
        atribui(vetor, i, i * 2);
    for(i = -29; i <= 30; i++)
        printf("vetor[%d] = %d\n", i, consulta(vetor, i));
    libera_vetor(vetor);
    return 0;
}
```

Exercícios

1. Implementar as operações sobre o TAD Ponto:
   - `void imprimir(Ponto *p)`
     - imprime as coordenadas de um ponto.
   - `void ler(Ponto *p)`
     - lê as coordenadas e cria um ponto.
   - `int eh_triang(Ponto *p1, Ponto *p2, Ponto *p3)`
     - verifica se 3 pontos formam um triângulo.

Exercícios

2. Usando o TAD Ponto, implemente os seguintes TADs:
   - Quadrado;
   - Círculo;
   - Triângulo;
   - Em todos, criar as operações:
     - cria;
     - libera;
     - acessa;
     - atribui;
     - verifica {verifica se o ponto está dentro ou fora do TAD}.

Exercícios

3. TAD Racional (site da monitoria):

```
struct racional
{
  int numerador;
  int denominador;
};

typedef struct racional Racional;
```

Exercícios

3. TAD Racional (site da monitoria):

- `Racional cria(int a, int b);`
- `Racional* soma(Racional *rac1, Racional *rac2);`
- `Racional* subtracao(Racional *rac1, Racional *rac2);`
- `Racional* multiplicacao(Racional *rac1, Racional *rac2);`
- `Racional* divisao(Racional *rac1, Racional *rac2);`
- `Racional* imprime(Racional *rac);`
- `int eh_igual(Racional *rac1, Racional *rac2);`
- `void libera(Racional *rac);`