

Tipo Abstrato de Dados

Hebert Coelho e Nádia Félix

Quando iniciamos nossos estudos em uma linguagem de programação, vários conceitos são apresentados. Um dos conceitos é o de **tipo de dados**.

Tipo de Dados

- Define um **conjunto de valores** (domínio) e **operações** que uma variável pode assumir.
- Exemplos:
 - **inteiro**: admite valores $\langle ? -2, -1, 0, +1, +2, ? \rangle$ e suporta operações de soma, subtração, etc.
 - **real**
 - **caractere**

Tipos de Dados

Podem ser classificados em:

- Primitivos ou básicos
- Estruturados
- Definidos pelo usuário

Tipos de dados Primitivos ou básicos



Tipos de dados estruturados (construídos)

- Uma estrutura de dados é uma forma de armazenar e organizar os dados de modo que eles possam ser usados de forma eficiente.

Tipos de dados estruturados (construídos)

- Uma estrutura de dados é uma forma de armazenar e organizar os dados de modo que eles possam ser usados de forma eficiente.
- Consiste em:
 - um conjunto de tipos de dados;
 - Definição de operações que podem ser realizadas sobre este conjunto de dados.

Tipos de dados estruturados (construídos)

- Uma estrutura de dados é uma forma de armazenar e organizar os dados de modo que eles possam ser usados de forma eficiente.
- Consiste em:
 - um conjunto de tipos de dados;
 - Definição de operações que podem ser realizadas sobre este conjunto de dados.
- Exemplos:
 - arranjos (vetores e matrizes);
 - estruturas (struct);
 - referências (ponteiros).

Tipos definidos pelo usuário

- Muitas vezes, os tipos de dados e as estruturas de dados presentes na linguagem podem não ser suficientes para nossa aplicação.

Tipos definidos pelo usuário

- Muitas vezes, os tipos de dados e as estruturas de dados presentes na linguagem podem não ser suficientes para nossa aplicação.
- Podemos necessitar de uma melhor estruturação dos dados, assim como especificar quais operações estarão disponíveis para manipular esses dados.

Tipos definidos pelo usuário

- Muitas vezes, os tipos de dados e as estruturas de dados presentes na linguagem podem não ser suficientes para nossa aplicação.
- Podemos necessitar de uma melhor estruturação dos dados, assim como especificar quais operações estarão disponíveis para manipular esses dados.
- Assim convém criar um tipo abstrato de dado, também conhecido como TAD.

Tipos e Estruturas de Dados

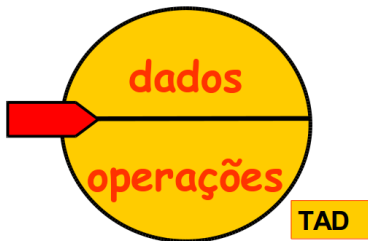
Tipos de dados básicos

Fornecidos pela Linguagem de Programação

Estruturas de Dados

- Estruturação conceitual dos dados;
- Reflete um relacionamento lógico entre dados, de acordo com o problema considerado.

Tipos e Estruturas de Dados



Um **TAD** é uma forma de definir um **novo tipo** de dado juntamente com as **operações** que manipulam esse novo tipo de dado.

TAD

TAD

- Separação entre conceito (definição do tipo) e implementação das operações;
- Visibilidade da estrutura interna do tipo fica limitada às operações;
- Aplicações que usam o TAD são denominadas clientes do tipo de dado;
- Cliente tem acesso somente à forma abstrata do TAD.

TAD

TAD

- Separação entre conceito (definição do tipo) e implementação das operações;
- Visibilidade da estrutura interna do tipo fica limitada às operações;
- Aplicações que usam o TAD são denominadas clientes do tipo de dado;
- Cliente tem acesso somente à forma abstrata do TAD.

TAD

TAD

- O TAD estabelece o conceito de **tipo de dado separado da sua representação.**

TAD

TAD

- O TAD estabelece o conceito de **tipo de dado separado da sua representação**.
- Definido como um modelo matemático por meio de um par (v, o) em que:
 - v é um conjunto de valores
 - o é um conjunto de operações sobre esses valores

TAD

TAD

- O TAD estabelece o conceito de **tipo de dado separado da sua representação**.
- Definido como um modelo matemático por meio de um par (v, o) em que:
 - v é um conjunto de valores
 - o é um conjunto de operações sobre esses valores
 - Ex.: tipo real
 - $v = \mathbb{R}$
 - $o = \{+, -, *, /, =, <, >, <=, >=\}$

Vantagens do TAD

- Código do cliente do TAD não depende da implementação

Vantagens do TAD

- Código do cliente do TAD não depende da implementação
- Segurança:

Vantagens do TAD

- Código do cliente do TAD não depende da implementação
- Segurança:
 - clientes não podem alterar a representação

Vantagens do TAD

- Código do cliente do TAD não depende da implementação
- Segurança:
 - clientes não podem alterar a representação
 - clientes não podem tornar os dados inconsistentes

Projeto de um TAD

- Envolve a escolha de operações adequadas para uma determinada estrutura de dados, definindo seu comportamento

Projeto de um TAD

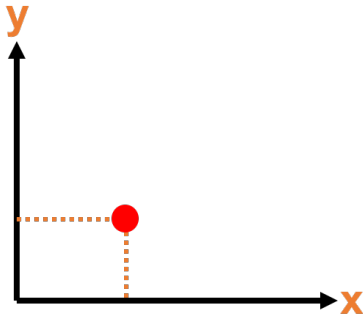
- Envolve a escolha de operações adequadas para uma determinada estrutura de dados, definindo seu comportamento
- **Dicas para definir um TAD:**
 - definir pequeno número de operações
 - conjunto de operações deve ser suficiente para realizar as computações necessárias às aplicações que utilizarem o TAD
 - cada operação deve ter um propósito bem definido, com comportamento constante e coerente

Exemplo de TAD:: representação de um ponto

- Ponto (x,y)
 - Coordenada x
 - Coordenada y

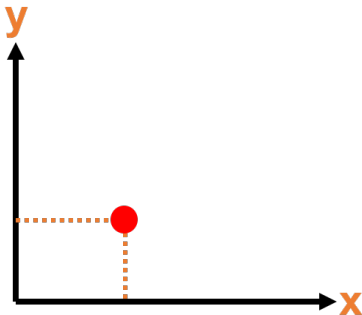
Exemplo de TAD:: representação de um ponto

- Ponto (x,y)
 - Coordenada x
 - Coordenada y



Exemplo de TAD:: representação de um ponto

- Ponto (x,y)
 - Coordenada x
 - Coordenada y
- Par (v,o) :
 - v – dupla formada por dois reais: $\text{Ponto}(x,y)$
 - o – operações aplicáveis sobre o tipo Ponto



Exemplo de TAD:: representação de um ponto

- Operações

- *pto_cria*: operação que cria um ponto, alocando memória para as coordenadas x e y;
- *pto_libera*: operação que libera a memória alocada por um ponto;
- *pto_acessa*: operação que devolve as coordenadas de um ponto;
- *pto_atribui*: operação que atribui novos valores às coordenadas de um ponto;
- *pto_distancia*: operação que calcula a distância entre dois pontos.

Modularização e Implementação do TAD

- A convenção em linguagem C é preparar dois arquivos para implementar uma TAD

Modularização e Implementação do TAD

- A convenção em linguagem C é preparar dois arquivos para implementar uma TAD
- **Arquivo .H** : protótipos das funções, tipos de ponteiro, e dados globalmente acessíveis. Aqui é definida a interface visível pelo usuário.

Modularização e Implementação do TAD

- A convenção em linguagem C é preparar dois arquivos para implementar uma TAD
- **Arquivo .H** : protótipos das funções, tipos de ponteiro, e dados globalmente acessíveis. Aqui é definida a interface visível pelo usuário.
- **Arquivo .C**: declaração do tipo de dados e implementação das suas funções. Aqui é definido tudo que ficará oculto do cliente da TAD.

Modularização e Implementação do TAD

- A convenção em linguagem C é preparar dois arquivos para implementar uma TAD
- **Arquivo .H** : protótipos das funções, tipos de ponteiro, e dados globalmente acessíveis. Aqui é definida a interface visível pelo usuário.
- **Arquivo .C**: declaração do tipo de dados e implementação das suas funções. Aqui é definido tudo que ficará oculto do cliente da TAD.
- Assim separamos o “conceito” (definição do tipo) de sua “implementação”.
- **A esse processo de separação da definição do TAD em dois arquivos damos o nome de modularização.**

Interface

- Eu não estou falando de Interface gráfica ok?
- Trata-se do Protótipo de função ou declaração de uma função
 - `int fac(int n);`
- Através da utilização de protótipos de função em arquivos de cabeçalho (normalmente, em programas escritos na linguagem C, em arquivos com a extensão “.h”) é possível especificar interfaces para bibliotecas de software.
- Na interface também podemos especificar tipos que são globais e portanto acessíveis globalmente
- E também podemos especificar ponteiros

Exemplo Ponto

- 1 Definir o arquivo “.H”
 - protótipos das funções
 - tipos de ponteiros
 - Dados globalmente acessíveis
 - Definir o arquivo “.C”
 - Na condição de cliente, usar...

Arquivo Ponto.h

```
1 //Arquivo Ponto.h
2 typedef struct ponto Ponto;
3
4 //Cria um novo ponto
5 Ponto* pto_cria(float x, float y);
6
7 //Libera um ponto
8 void pto_libera(Ponto* p);
9
10 //Acessa os valores "x" e "y" de um ponto
11 void pto_acessa(Ponto* p, float* x, float* y);
12
13 //Atribui os valores "x" e "y" de um ponto
14 void pto_atribui(Ponto* p, float x, float y);
15
16 //Calcula a distancia entre dois pontos
17 float pto_distancia(Ponto* p1, Ponto* p2);
18
19
```

Arquivo Ponto.c

```
1 //Arquivo Ponto.C
2 #include <stdlib.h> // Nós vamos usar a constante NULL
3 #include <math.h> // Nós vamos utilizar o calculo da distancia
4 #include "Ponto.h" //inclui os protótipos
5
6 //Definição de tipo de dados
7 struct ponto{
8     float x;
9     float y;
10 };
```

Continuação do Arquivo Ponto.c

```
11
12 //Arquivo Ponto.C
13 Ponto* pto_cria(float x, float y){
14     Ponto* p = (Ponto*) malloc(sizeof(Ponto));
15     if(p!=NULL){
16         p->x = x;
17         p->y = y;
18     }
19     return p;
20 }
21
22 //Libera a memória alocada para um ponto
23 void pto_libera(Ponto* p){
24     free(p);
25 }
```

Continuação do Arquivo Ponto.c

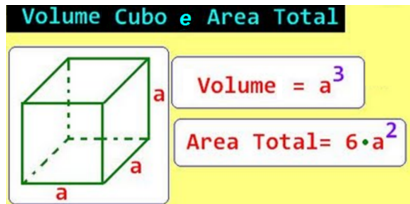
```
26
27 //Acessa os valores "x" e "y" de um ponto
28 void pto_acessa(Ponto* p, float* x, float* y){
29     *x = p->x;
30     *y = p->y;
31 }
32
33 //Atribui os valores "x" e "y" de um ponto
34 void pto_atribui(Ponto* p, float x, float y){
35     p->x = x;
36     p->y = y;
37 }
38
39 //Calcula a distancia entre dois pontos
40 float pto_distancia(Ponto* p1, Ponto* p2){
41     float dx = p1->x - p2->x;
42     float dy = p1->y - p2->y;
43     return sqrt(dx*dx+dy*dy);
44 }
45
46
47
```

Programa Cliente ? que usa o TAD

```
Ponto.h x Ponto.c x main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "Ponto.h"
4
5  int main()
6  {
7      float d;
8      Ponto *p,*q;
9      //Ponto r; // Erro
10     p = pto_cria(10,21);
11     q = pto_cria(7,25);
12     //q->X = 2; //ERRO
13     d = pto_distancia(p,q);
14     printf("Distancia entre pontos: %f\n",d);
15     pto_libera(p);
16     pto_libera(q);
17     return 0;
18 }
19
```

Exercícios

- Desenvolva um TAD que represente um cubo. Inclua as funções de inicialização necessárias e as operações que retornem o tamanho de cada lado, a sua área e o seu volume.



Cubo.h

```
1 //Arquivo Cubo.h
2 typedef struct cubo Cubo;
3
4 //Cria um novo cubo
5 Cubo* cubo_cria(float a);
6
7 //Libera um cubo
8 void cubo_libera(Cubo* c);
9
10 //Acessa o valor "a" de Cubo
11 float cubo_acessa(Cubo* c);
12
13 //Atribui o valor a de um Cubo
14 void cubo_atribui(Cubo* c, float a);
15
16 //Calcula a area total de Cubo
17 float cubo_area(Cubo* c);
18
19 //Calcula o volume de um Cubo
20 float cubo_volume(Cubo* c);
21
```

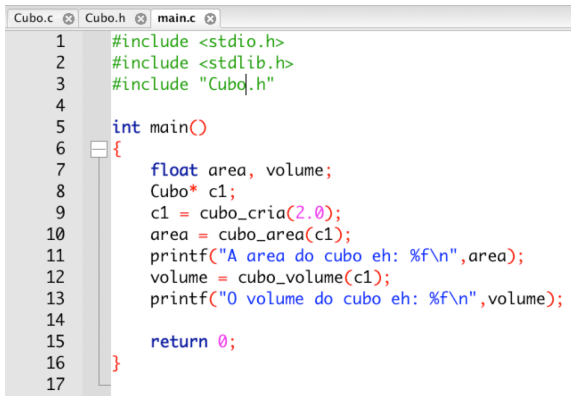

Cubo.c

```
1  #include <stdlib.h> // Nós vamos usar a constante NULL
2  #include "Cubo.h" //inclui os protótipos
3
4  //Definição do tipo de dados
5  struct cubo{
6      float a;
7  };
8
9  //Cria um novo cubo
10 Cubo* cubo_cria(float a){
11     Cubo* c = (Cubo*) malloc(sizeof(Cubo));
12     if(c!=NULL){
13         c->a = a;
14     }
15     return c;
16 }
17
18 //Libera um cubo
19 void cubo_libera(Cubo* c){
20     free(c);
21 }
22
```

Cubo.c - Continuação

```
23 //Acessa o valor "a" de Cubo
24 float cubo_acessa(Cubo* c){
25     return c->a;
26 }
27
28 //Atribui o valor a de um Cubo
29 void cubo_atribui(Cubo* c, float a){
30     c->a = a;
31 }
32
33 //Calcula a area total de Cubo
34 float cubo_area(Cubo* c){
35     float area = 6 * c->a * c->a;
36     return area;
37 }
38
39 //Calcula o volume de um Cubo
40 float cubo_volume(Cubo* c){
41     float volume = c->a * c->a * c->a;
42     return volume;
43 }
44
```

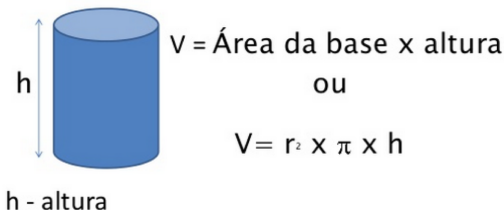
Programa Cliente – que usa o TAD



```
Cubo.c x Cubo.h x main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "Cubo.h"
4
5  int main()
6  {
7      float area, volume;
8      Cubo* c1;
9      c1 = cubo_cria(2.0);
10     area = cubo_area(c1);
11     printf("A area do cubo eh: %f\\n",area);
12     volume = cubo_volume(c1);
13     printf("O volume do cubo eh: %f\\n",volume);
14
15     return 0;
16 }
17
```

Exercícios

- Desenvolva um TAD que represente um cilindro. Inclua as funções de inicializações necessárias e as operações que retornem a sua altura, o raio, e seu volume. Os arquivos devem ser zipados e enviados pelo SIGAA.



Exercícios

- Desenvolva um TAD que represente uma esfera. Inclua as funções de inicializações necessárias e as operações que retornem o seu raio, a sua área e o seu volume. Os arquivos devem ser zipados e enviados pelo SIGAA.

