TADs
Tipos Abstratos de Dados

Wanderley de Souza Alencar
wanderleyalencar@ufg.br

Universidade Federal de Goiás - UFG
Instituto de Informática - INF

29 de março de 2020

Introdução

Introdução

...

Normalmente quando se inicia o estudo de *algoritmos* e/ou de *linguagens de programação*, vários conceitos são apresentados.

Um destes conceitos é o de dado.

O que é mesmo um *dado*?

Introdução

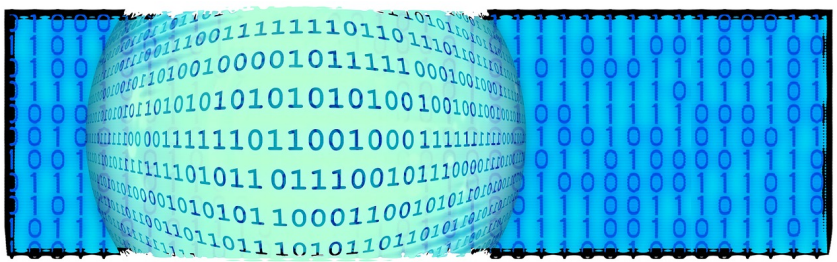
Dado – Conceito

A palavra *dado* indica a representação de uma certa informação.

A informação é o resultado do tratamento dos dados que podem estar sob a forma de um texto, imagem, vídeo, áudio, etc.

No âmbito da Computação, dados são processados pela CPU do computador e armazenados em memória principal ou qualquer outro dispositivo de armazenamento secundário.

No nível mais elementar, os dados são simplesmente *cadeias binárias*.



Introdução

...

Outro conceito importante é o de tipo de dado.

O que é um *tipo de dado*?

Introdução

Tipo de Dado – Conceito

Um *tipo de dado* define um conjunto de possíveis valores que uma certa variável pode assumir, bem como o conjunto de operações típicas, normalmente predefinidas, que podem ser realizadas com a variável (ou sobre a variável).

Introdução

Tipo de Dado – Conceito

Por exemplo:

Se a variável X é do tipo `inteiro`, então espera-se que ela possa assumir valores no conjunto:

$$\mathbb{Z} = \{\ldots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \ldots\}$$

E que estejam disponíveis as opeações de adição (+), subtração (-), multiplicação (x) e divisão inteira (/), pois elas são normalmente consideradas *operações típicas* sobre os números inteiros da Matemática.

Introdução

Tipo de Dado – Conceito

Numa linguagem de programação específica, um mesmo tipo de dado pode ter o par (`valores`, `operações`) variando de acordo com o ambiente operacional em que ela está implementada.

Introdução

Tipo de Dado – Conceito

Por exemplo:

No ambiente operacional A, a implementação dos inteiros para uma certa linguagem de programação $\mathbb{X}$ pode permitir números inteiros na faixa:

-2^{31} a $(2^{31} - 1)$

Já no ambiente operacional B, os inteiros de $\mathbb{X}$ estão na faixa:

-2^{63} a $(2^{63} - 1)$.

Introdução

Tipo de Dado – Conceito

Enfatizando...

Um *tipo de dado* definirá um conjunto de

- valores (domínio) e de
- operações

associados a uma variável qualquer daquele tipo.

Introdução

Tipo de Dado – Exemplo

Exemplo: **inteiro**

- domínio:
  - $< ? \ -2, -1, 0, +1, +2, \ ? >$
- operações:
  - adição;
  - subtração;
  - multiplicação;
  - divisão inteira;
  - ...

Introdução

Tipo de Dado – Exemplo

Exemplo: **caractere**

- domínio
  - valores definidos por uma *tabela* de codificação (ASCII, Unicode, etc.).
- operações
  - comparação ($<, >, \leq, \geq, \neq, \ldots$);
  - conversão maiúscula/minúscula;
  - ...

Introdução

Tipo de Dado – Exemplo

Tabela ASCII

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Introdução

Tipo de Dado – Exemplo

Tabela UNICODE

`https://pt.wikipedia.org/wiki/Unicode`

Introdução

Tipo de Dado – Classificação

Nas linguagens de programação contemporâneas, normalmente um tipo de dado pode ser classificado como:

- Primitivo (básico, nativo, fundamental ou elementar);
- Estruturado (ou composto);
- Definido pelo usuário (ou construído).

Introdução

Tipo de Dado – Primitivo

O tipo primitivo é aquele fornecido pela própria definição/implementação da linguagem de programação.

Normalmente reflete elemento presentes no próprio *hardware* subjacente ao ambiente computacional no qual a linguagem está implementada. Algumas vezes incorporaram algum nível (pequeno) de abstração em relação ao *hardware*.

Portanto, os tipos primitivos formam um conjunto básico de tipos de dados disponíveis para o programador daquela linguagem.

Introdução

Tipo de Dado – Primitivo

São *normalmente* tipos primitivos:

- inteiro;
- real (número em *ponto flutuante*);
- complexo;
- caractere;
- lógico;
- ponteiro (ou apontador);
- ...

Introdução

Tipo de Dado – Primitivo

Por exemplo, em $\mathbb{C}$:

- `short int` e `unsigned short int`;
- `int` e `unsigned int`;
- `long int` e `unsigned long int`;
- `long long int` e `unsigned long long int`;
- `float`;
- `double` e `long double`;
- `char` e `unsigned char`.

Introdução

Tipo de Dado – Estruturado

O tipo estruturado é formado a partir da composição, ou agregação, de tipos primitivos ou de outros tipos estruturados previamente definidos.

Exige, por parte da linguagem que o possui, maior nível de abstração para sua implementação.

Ele permite, de maneira *elegante*, organizar os dados de modo que possam ser utilizados eficientemente, reunindo um conjunto de vários tipos de dados sob um único conceito.

Introdução

Tipo de Dado – Estruturado

São *normalmente* tipos estruturados:

- cadeia de caracteres, ou *string*;
- registro;
- vetor;
- matriz;
- ...

Introdução

Tipo de Dado – Estruturado

Por exemplo, em $\mathbb{C}$:

- `struct`;
- `union`.

Introdução

Tipo de Dado – Estruturado

Exemplo em $\mathbb{C}$:

```
1  //
2  // Representacao de uma conta-corrente numa instituicao financeira
3  //
4  struct ContaCorrente {
5      unsigned int numero;
6      char * nomeTitular;
7      unsigned int telefoneTitular;
8      bool contaConjunta;
9      char * nomeDependente;
10     float saldoAtual;
11     bool estaAtiva;
12 }
```

Introdução

Tipo de Dado – Estruturado

Exemplo em $\mathbb{C}$:

```c
//
// Representacao uma unica variavel que
// representa multipos tipos de dados.
//
union Data {
   int intN;
   float floatF;
   char str[20];
} data;

int main( ) {
   union Data data;

   printf("Memoria ocupada: %d\n", sizeof(data));
   return (0);
}
```

Introdução

Tipo de Dado – Definido pelo Usuário

Os tipos de dados presentes numa linguagem de programação podem não ser suficientes para o desenvolvimento de uma certa aplicação.

Assim convém termos a possibilidade de criar novos tipos de dados que, por consequência, são chamados de tipos definidos pelo usuário.

Introdução

Tipo de Dado – Definido pelo Usuário

Por exemplo, em $\mathbb{C}$:

- `enum`;
- `bool`;

Introdução

Tipo de Dado – Definido pelo Usuário

Exemplo em $\mathbb{C}$:

```c
#include <stdio.h>
//
// Declaracao de uma enumeracao
//
enum DiaSemana {Domingo, Segunda, Terca, Quarta, Quinta, Sexta, Sabado};

int main( ) {
    enum DiaSemana dia;

    dia = Quinta;
    if (dia == Domingo) {
        printf("FDS\n");
    }
    else {
        printf("%d\n", dia);
    }
    return (0);
}
```

Introdução

Tipo de Dado – Definido pelo Usuário

Exemplo em $\mathbb{C}$:

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  //
4  // Declaracao de um tipo logico (boolean)
5  //
6  int main( ) {
7     bool emCrash;
8
9     emCrash = false;
10    if (emCrash == true) {
11       printf("Sistema em CRASH.\n");
12    }
13    else {
14       printf("Sistema OK!\n");
15    }
16    return (0);
17 }
```

Introdução

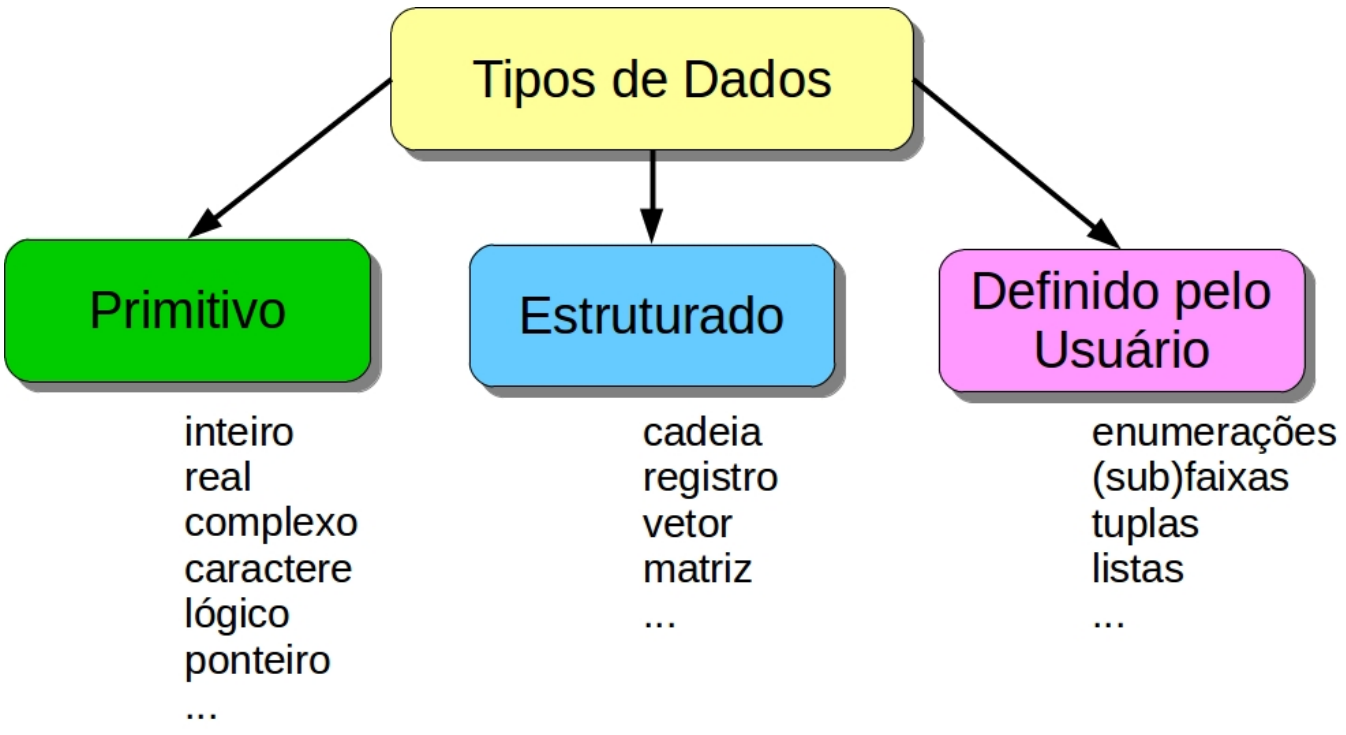
Tipo de Dado – Definido pelo Usuário

Há autores que consideram que, em $\mathbb{C}$, os tipos `struct` e `union` são também tipos definidos pelo usuário.

Em minha opinião eles são *tipos estruturados* e, por isso, assim os classifiquei.

Portanto ficará a seu critério adotar uma ou outra classificação.

Tipo de Dado – Definido pelo Usuário



TAD – Tipo Abstrato de Dado

TAD – Tipo Abstrato de Dado

...

Em muitas situações concebemos, mentalmente, novos *tipos de dados* e imaginamos *operações* sendo realizadas com eles, ou sobre eles.

Assim convém termos a possibilidade de criar novos tipos de dados, como também especificar quais serão as operações disponíveis para manipular variáveis destes *novos* tipos.

TAD – Tipo Abstrato de Dado

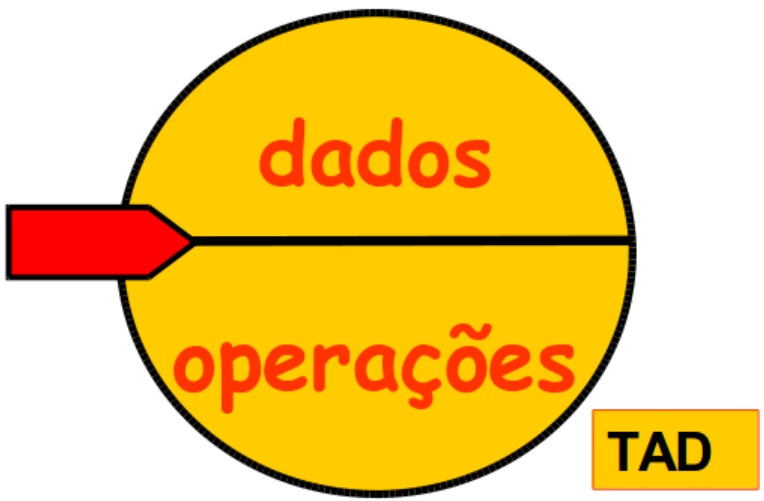
...

Estes *novos tipos*, com as suas *operações*, são chamados de **T**ipos **A**bstratos de **D**ados – TADs.

TAD – Tipo Abstrato de Dado

TAD – Conceito

Um TAD é uma maneira do programador definir um *novo tipo* de dados juntamente com as *operações* que manipularão este novo tipo.



TAD – Tipo Abstrato de Dado

O que é *abstração*?

"*Uma abstração é a visão, ou representação, de uma entidade que inclui apenas seus atribuitos mais significativos. No sentido geral, a abstração permite-nos colecionar instâncias de entidades em grupos nos quais seus atributos comuns não precisam ser considerados.*"

(SEBESTA, *Concepts of Programming Languages*, 2012, pp. 474).

TAD – Tipo Abstrato de Dado

O que é *abstração*?

A *abstração*, numa linguagem de programação, é uma arma contra a complexidade da própria programação, simplificando este processo.

Com o uso da *abstração* é possível concentrar-nos nos aspectos essenciais de um contexto qualquer, ignorando características que são somenos importantes, acidentais, acessórias.

TAD – Tipo Abstrato de Dado

O que é *abstração*?

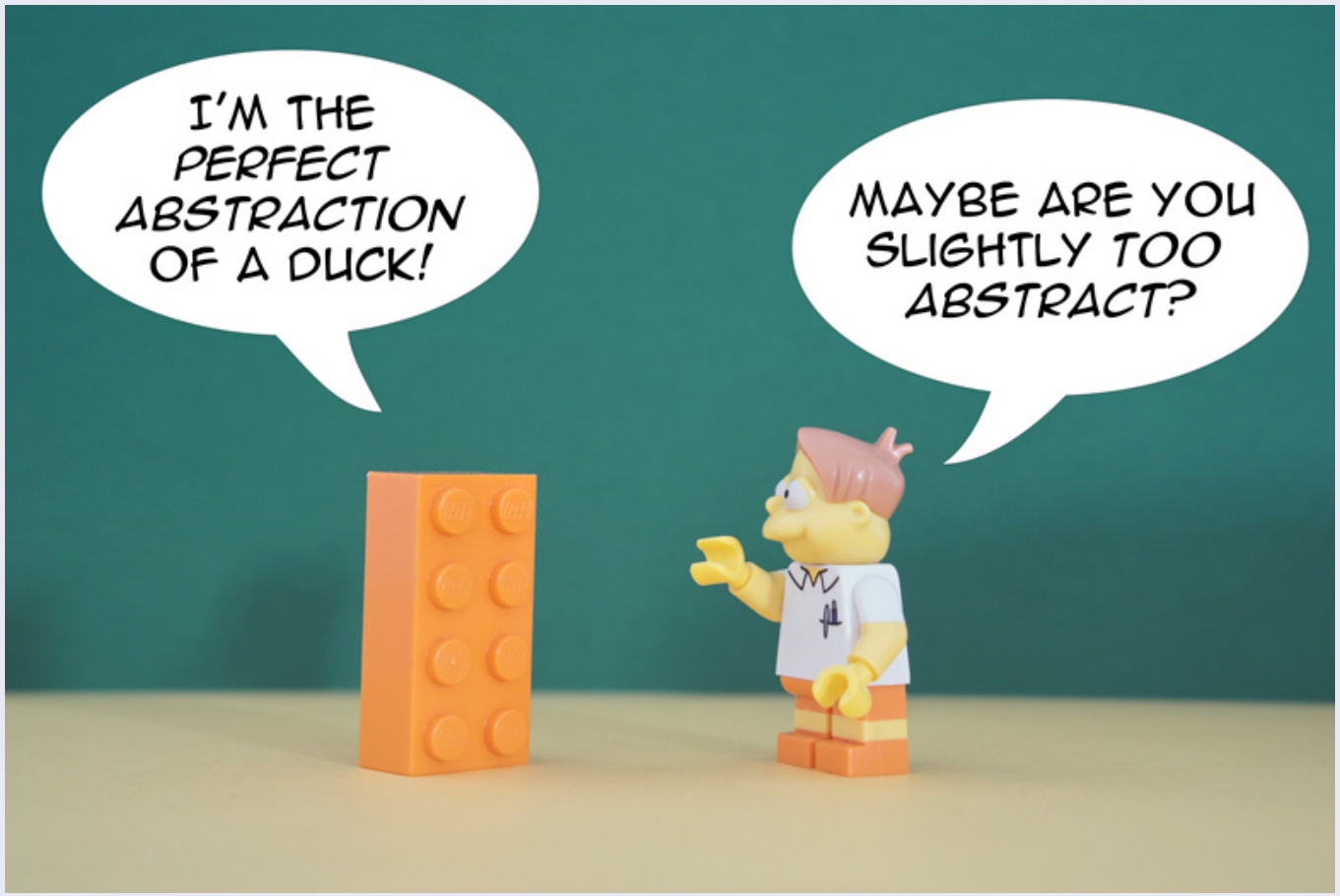


Figura: from The Valuable Dev.

TAD – Tipo Abstrato de Dado

O que é *abstração*?

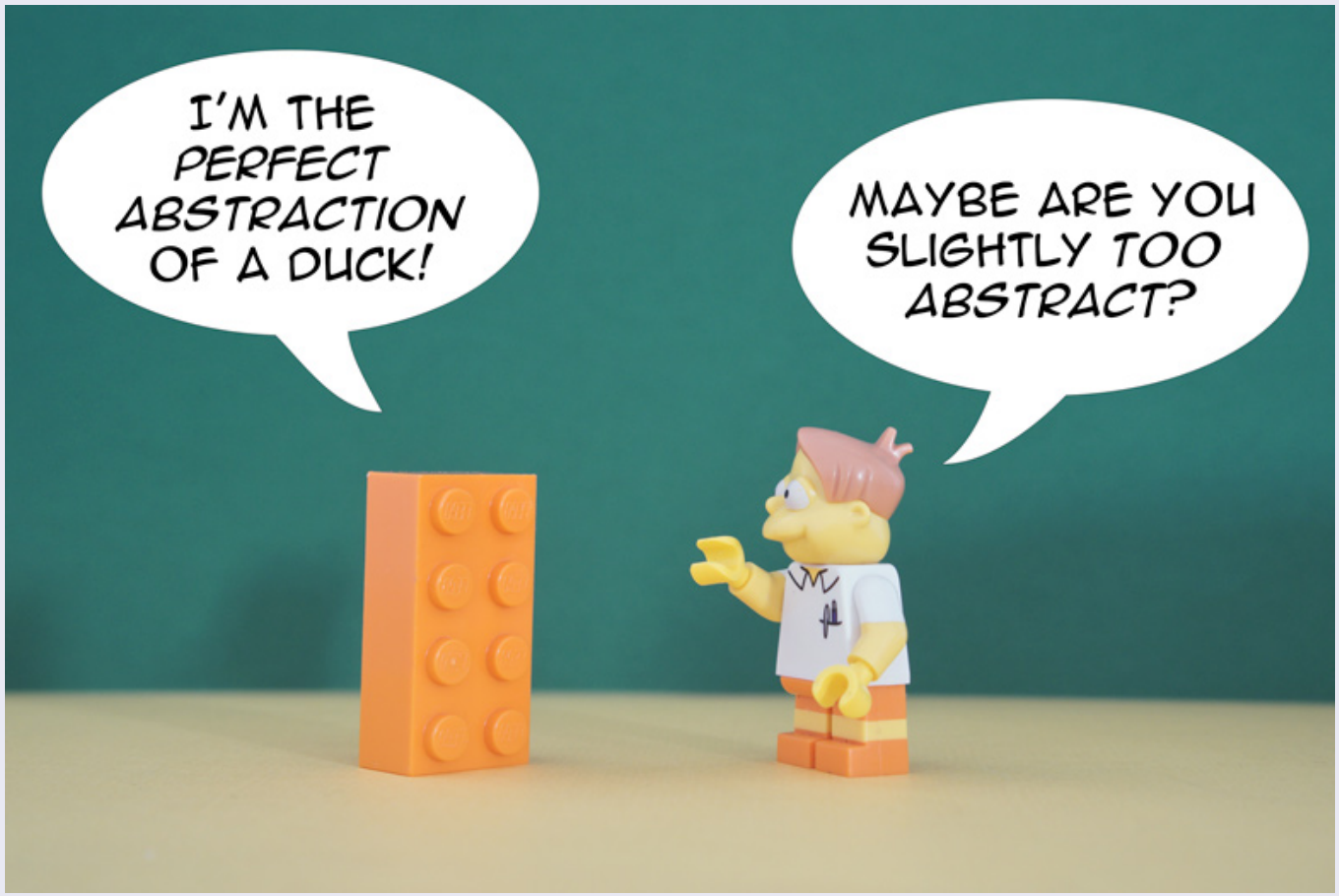


Figura: from The Valuable Dev.

TAD – Tipo Abstrato de Dado

TAD – Conceito

Um Tipo Abstrato de Dado (TAD) é a especificação de um conjunto de dados e das operações que podem ser executadas sobre eles, independentemente da maneira como tudo isto será implementado.

O objetivo é proporcionar ao desenvolvedor uma *abstração* que facilite a concepção – e programação – de uma aplicação, pois as variáveis de um TAD podem ser tratadas como "*entidades fechadas*" (ou *caixas-pretas*).

TAD – Tipo Abstrato de Dado

Figura: Visão de *caixa preta*.

TAD – Tipo Abstrato de Dado

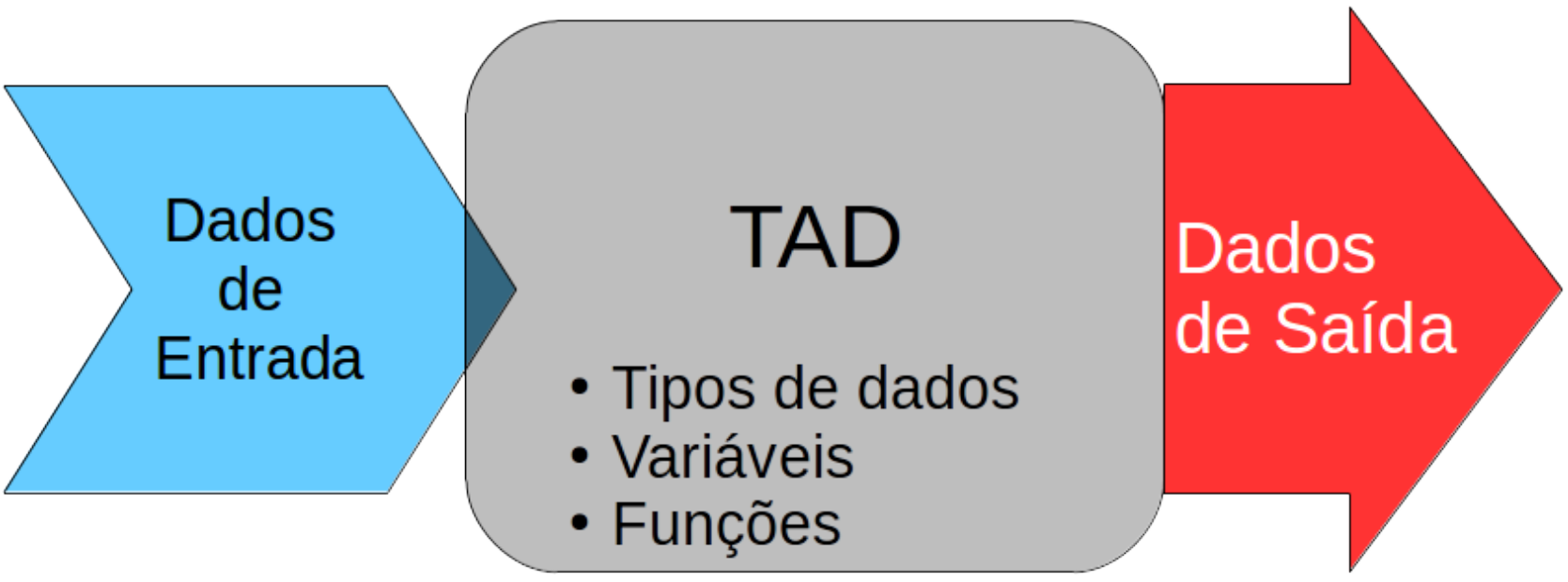


Figura: Visão INTERNA da *caixa preta*.

TAD – Tipo Abstrato de Dado

Conceito

Um TAD...

- estabelece o conceito de tipo de dado separado de sua representação.

- é definido como um modelo matemático por meio de um par (`valores`, `operações`) em que:
  - valores – conjunto de valores que podem ser assumidos por uma variável daquele tipo;

  - operações – conjunto de operações possíveis sobre uma variável daquele tipo.

TAD – Tipo Abstrato de Dado

TAD – Exemplo 01

O tipo *números reais* – $\Re$ – pode ser definido como um TAD por:

- `valores` – $\Re$
- `operações` – $\{+, -, *, /, =, \neq, <, \leq, >, \geq\}$

TAD – Tipo Abstrato de Dado

Vantagens

A definição de um TAD, permite:

- separação entre conceito (definição do tipo) e implementação das operações;
- limitar a visibilidade da estrutura interna do TAD;
- controlar a visibilidade das operações perante o usuário, que passa a ser cliente do TAD;
- limitar o acesso do cliente somente à forma abstrata do TAD.
- ...

TAD – Tipo Abstrato de Dado

Vantagens

A definição de um TAD, permite:

- que o código do cliente não dependa da implementação dele: o TAD é uma *caixa preta* sob a ótica do cliente;
- segurança: clientes não podem alterar a representação, pois não possuem acesso a ela;
- segurança: clientes não podem tornar os dados inconsistentes, pois não possuem acesso a eles.

TAD – Tipo Abstrato de Dado

Projeto

Algumas diretrizes para projetar um *bom* TAD são:

- escolher as operações adequadas, definindo claramente o comportamento de cada uma delas;
- projetar operações flexíveis e suficientemente abrangentes para os diversos contextos de uso do TAD;
- implementar eficientemente cada operação definida;
- reutilizar operações básicas para elaborar outras mais complexas.

TAD – Tipo Abstrato de Dado

Projeto

Escolher as operações adequadas significa:

- definir um *pequeno* número de operações;
- que o conjunto de operações deve ser suficiente para realizar as computações necessárias às aplicações que utilizarão o TAD;
- que cada operação deve ter um propósito bem definido, com comportamento constante e coerente.

TAD – Tipo Abstrato de Dado

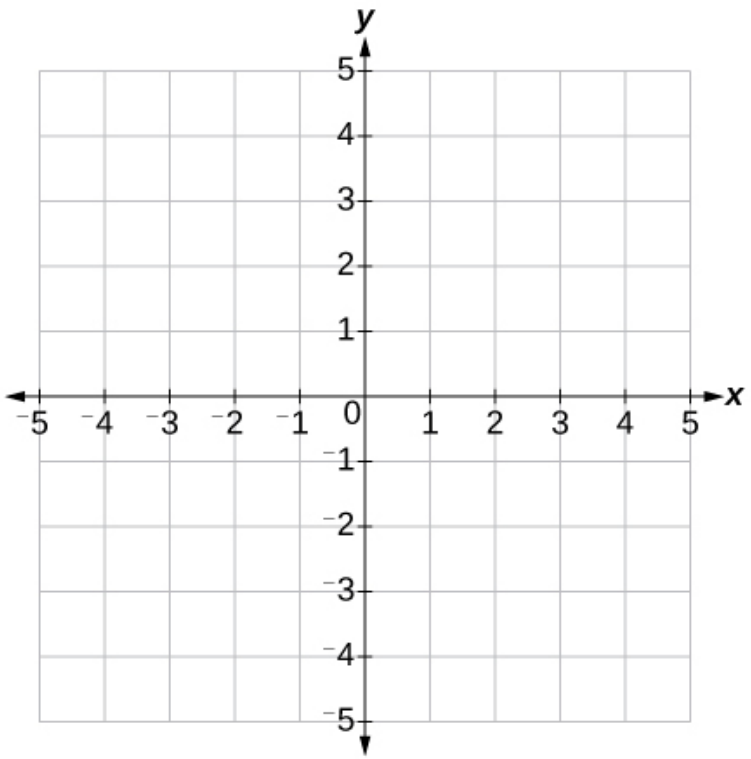
TAD – Exemplo 02

Apenas um Exemplo

TAD – Exemplo 02

Ponto 2D

Considere que numa certa aplicação da área de *Computação Gráfica* deseja-se ter um TAD que represente um *ponto* no espaço bidimensional.



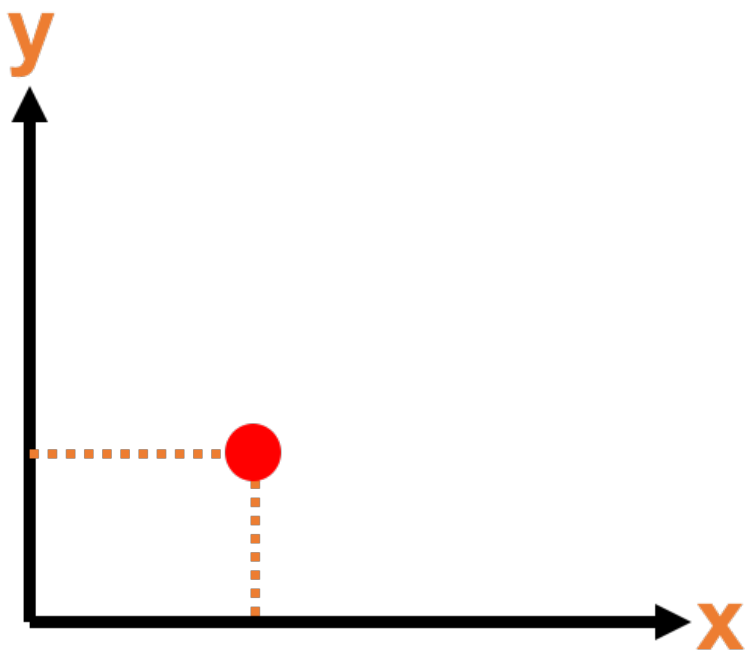
TAD – Exemplo 02

Ponto 2D

Matematicamente um *ponto* no espaço bidimensional possui:

- Coordenada x – um número real ($x \in \Re$);
- Coordenada y – um número real ($y \in \Re$).

TAD – Exemplo 02



TAD – Exemplo 02

- Ponto (x,y)
  - Coordenada x – um número real ($x \in \Re$);
  - Coordenada y – um número real ($y \in \Re$).

- Par (v,o):
  - `valores` – dupla ordenada formada por dois reais: Ponto(x,y);
  - `operações` – operações aplicáveis sobre o tipo Ponto.

TAD – Exemplo 02

Operações

Quais operações são necessárias, neste contexto, em relação a um Ponto2D?

TAD – Exemplo 02

Operações

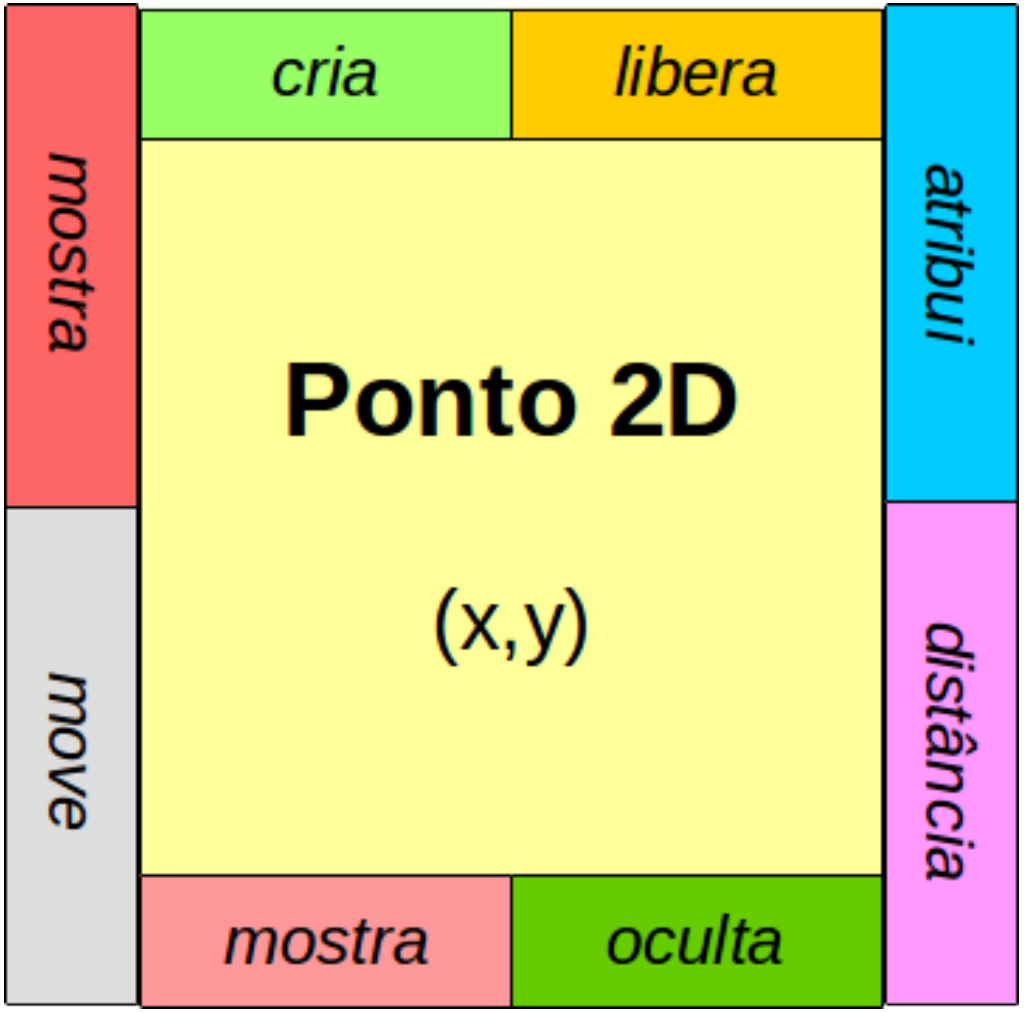
- *ponto_cria*: cria um ponto, alocando memória para as suas coordenadas;
- *ponto_libera* : libera a memória alocada por um ponto, destruindo-o;
- *ponto_acessa* : retorna as coordenadas de um ponto;
- *ponto_atribui* : atribui novos valores às coordenadas de um ponto;
- ...

TAD – Exemplo 02

Operações

- *ponto_distancia* : calcula a distância *euclidiana* entre dois pontos dados;
- *ponto_move*: move o ponto de uma posição para outra;
- *ponto_oculta* : torna o ponto *invisível*;
- *ponto_mostra* : torna o ponto *visível*;
- ...

TAD – Exemplo 02



TAD – Exemplo 02

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
//
// Definindo o tipo de dado
//

struct ponto {
    float x;
    float y;
    bool visibilidade; // true = visivel, false = invisivel
};
```

TAD – Exemplo 02

```
1  //
2  // Cria um ponto
3  //
4  Ponto* ponto_cria (float x, float y, bool visibilidade) {
5          Ponto* p = (Ponto*) malloc(sizeof(Ponto));
6          if (p != NULL) {
7                  p->x = x;
8                  p->y = y;
9                  p->visibilidade = visibilidade;
10         }
11         return (p);
12 }
```

TAD – Exemplo 02

```
1 //
2 // Libera (desaloca) um ponto...
3 //
4 void ponto_libera (Ponto* p) {
5     if (p != NULL) {
6         free(p);
7     }
8 }
```

TAD – Exemplo 02

```
1 //
2 // Acessa um ponto, coletando suas coordenadas
3 //
4 void ponto_acessa (Ponto* p, float* x, float* y) {
5     if (p != NULL) {
6             *x=p->x;
7             *y=p->y;
8         }
9 }
```

TAD – Exemplo 02

```
1 //
2 // Atribui coordenadas a um ponto, modificando-o
3 //
4 void ponto_atribui (Ponto* p, float x, float y) {
5     if (p != NULL) {
6         p->x=x;
7         p->y=y;
8     }
9 }
```

TAD – Exemplo 02

```
1  //
2  // Retorna a distancia entre dois pontos
3  //
4  float ponto_distancia (Ponto* p1, Ponto* p2) {
5          float dx, dy;
6
7          dx = p1->x - p2->x;
8          dy = p1->y - p2->y;
9          return (sqrt(dx*dx+dy*dy));
10 }
```

TAD – Exemplo 02

```
1  //
2  // move o ponto para as coordenadas (x, y)
3  //
4  void ponto_move (Ponto* p, float x, float y, int movimento) {
5
6        //
7        // Move o ponto p, a partir de sua posicao atual, para
8        // a posicao (x, y) segundo um certo tipo de movimento.
9        //
10       // Por exemplo: 1 - linear
11       // 2 - zig-zag
12       // ...
13       //
14
15 }
```

TAD – Exemplo 02

```
1 //
2 // Oculta (torna invisivel) o ponto
3 //
4 void ponto_oculta (Ponto* p) {
5
6     p->visibilidade = false;
7 }
```

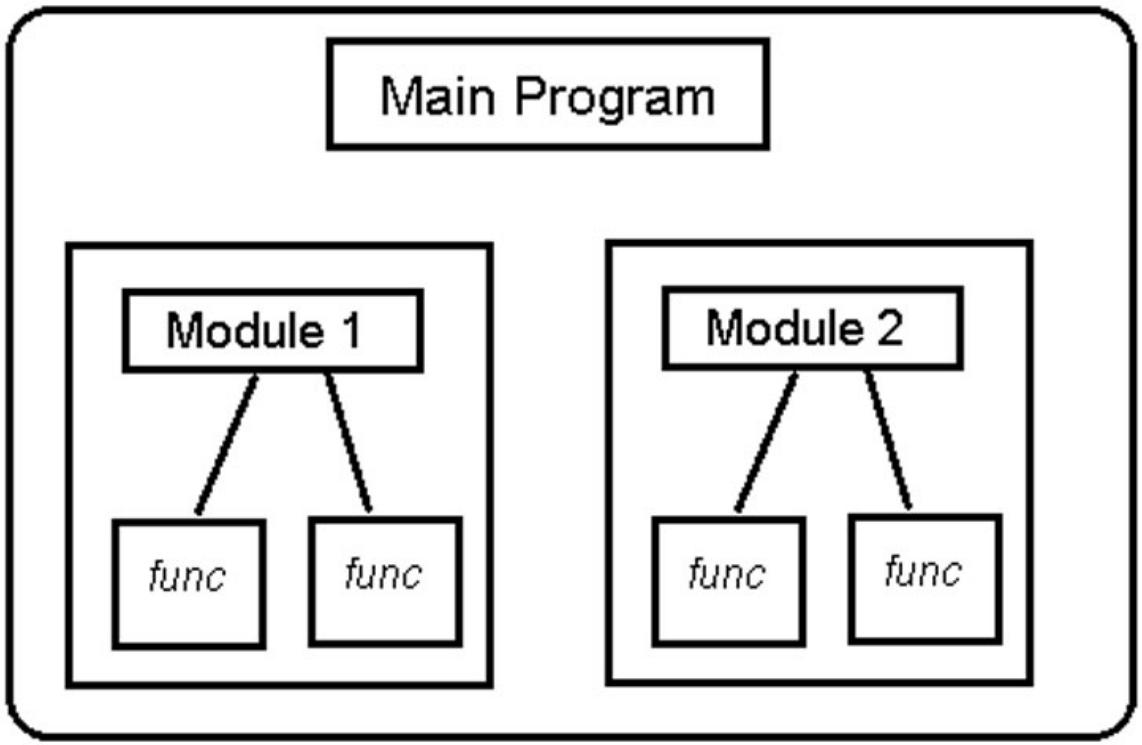
TAD – Exemplo 02

```
1 //
2 // Mostra (torna visivel) o ponto
3 //
4 void ponto_mostra (Ponto* p) {
5
6     p->visibilidade = true;
7 }
```

TAD – Exemplo 02

```
1  int main(){
2          float xp,yp,xq,yq,d;
3          Ponto *p,*q;
4
5          printf("digite as coordenadas x e y para o ponto 1: ");
6          scanf("%f %f",&xp,&yp);
7          printf("digite as coordenadas x e y para o ponto 2: ");
8          scanf("%f %f",&xq,&yq);
9          p = ponto_cria(xp,yp,true);
10         q = ponto_cria(xq,yq,true);
11         d = ponto_distancia(p,q);
12         ponto_acessa(p,&xp,&yp); ponto_acessa(q,&xq,&yq);
13         printf("Distancia entre os pontos (%.2f,%.2f) e (%.2f,%.2f) = %.5f\n",xp,
               yp,xq,yq,d);
14         ponto_libera(p); ponto_libera(q);
15         return (0);
16 }
```

TAD – Modularização e Implementação



TAD – Modularização e Implementação

Conceito

Sabemos que a definição de um TAD é conceitual: não há imposição quanto à implementação.

Cada linguagem de programação possui seus próprios padrões de como se implementar um TAD.

TAD – Modularização e Implementação

Conceito

Sabemos, ainda, que num TAD:

- a definição do tipo de dado (ou sua representação) e de suas operações são contidas numa única unidade sintática;
- a representação deve ocultar detalhes da implementação, exibindo apenas as operações que estão disponíveis para manipular aquele tipo de dado.

TAD – Modularização e Implementação

Conceito

Vamos, agora, nos concentrar na modularização e implementação de um TAD em...

TAD – Modularização e Implementação

Linguagem $\mathbb{C}$

Para criar um TAD na linguagem $\mathbb{C}$, convenciona-se preparar dois arquivos distintos:

- `tad.h`[a]
- `tad.c`

[a] O nome do arquivo pode ser livremente escolhido, desde que se utilize o mesmo nome para o arquivo **.h** e **.c**.

TAD – Modularização e Implementação

Linguagem $\mathbb{C}$

Na linguagem $\mathbb{C}$, convenciona-se preparar dois arquivos distintos:

- tad.h contém os *protótipos* das rotinas (procedimentos e funções), dos tipos ponteiro e de dados globalmente acessíveis à aplicação que utilizará o TAD.

Neste arquivo é definida a *interface* visível para o *cliente* do TAD.

TAD – Modularização e Implementação

Linguagem $\mathbb{C}$

Na linguagem $\mathbb{C}$, convenciona-se preparar dois arquivos:

- tad.c contém a declaração do tipo de dados e implementação das suas operações (procedimentos e funções).

O que for definido neste arquivo ficará invisível para o *cliente* do TAD.

TAD – Modularização e Implementação

Linguagem $\mathbb{C}$

- Dessa maneira 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.

TAD – Interface

...

É frequente, em Computação, que o termo `interface` se refira ao meio que o usuário utiliza para interagir com um sistema computacional:

- interface textual;
- interface gráfica;
- interface natural (por voz, por exemplo);
- ...

TAD – Interface

...

No contexto dos TADs, o termo `interface` refere-se ao *protótipo* de uma função, ou seja, à declaração de uma função:

- `int factorial(int n);`
- `void swap(int * a; int * b);`
- `int * allocArray(int * v; int n);`
- `...`

TAD – Interface

...

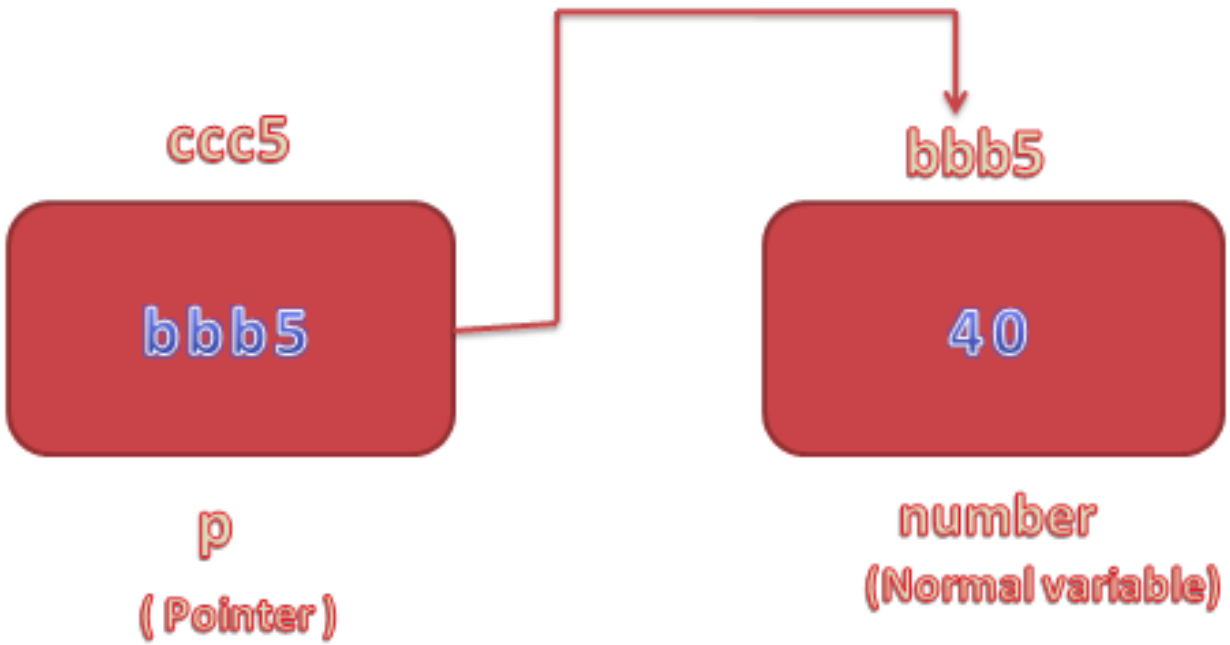
Por meio do uso de protótipos de função em arquivos de cabeçalho é possível especificar interfaces para bibliotecas de *software*.

Na interface pode-se especificar tipos que são globais e, portanto, acessíveis em todo o escopo da aplicação/módulo.

TAD – Interface

...

Na interface se pode especificar ponteiros.



TAD – Exemplo 03

Ponto 2D

Vamos (re)definir o *ponto 2D* por meio de uma TAD...

(1) definir o arquivo `tad.h`:
- protótipos das funções;
- tipos de ponteiros;
- dados globalmente acessíveis.

(2) definir o arquivo `tad.c`

(3) na condição de cliente, usar...

TAD – Exemplo 03

```
1  //
2  // Arquivo: ponto.h
3  //
4  typedef struct ponto Ponto;
5
6  Ponto* ponto_cria(float x, float y, bool visibilidade);
7
8  void ponto_libera(Ponto* p);
9
10 void ponto_acessa(Ponto* p, float* x, float* y);
11
12 void ponto_atribui(Ponto* p, float x, float y);
13
14 float ponto_distancia(Ponto* p1, Ponto* p2);
15
16 void ponto_oculta(Ponto* p);
17
18 void ponto_mostra(Ponto* p);
19
20 void ponto_move(Ponto* p, float x, float y);
```

TAD – Exemplo 03

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <stdbool.h>
5  #include "ponto.h" // inclusao do arquivo de cabecalho!
6
7  struct ponto {
8      float x;
9      float y;
10     bool visibilidade; // true = visivel, false = invisivel
11 };
12 //
13 // Aqui estara o codigo-fonte das implementacoes das rotinas.
14 //
```

TAD – Exemplo 03

```
1  //
2  // Cria um ponto
3  //
4  Ponto* ponto_cria (float x, float y, bool visibilidade) {
5          Ponto* p = (Ponto*) malloc(sizeof(Ponto));
6          if (p != NULL) {
7                  p->x = x;
8                  p->y = y;
9                  p->visibilidade = visibilidade;
10         }
11         return (p);
12 }
```

TAD – Exemplo 03

```
1 //
2 // Libera (desaloca) um ponto...
3 //
4 void ponto_libera (Ponto* p) {
5     if (p != NULL) {
6         free(p);
7     }
8 }
```

TAD – Exemplo 03

```
1 //
2 // Acessa um ponto, coletando suas coordenadas
3 //
4 void ponto_acessa (Ponto* p, float* x, float* y) {
5     if (p != NULL) {
6             *x=p->x;
7             *y=p->y;
8         }
9 }
```

TAD – Exemplo 03

```
1 //
2 // Atribui coordenadas a um ponto, modificando-o
3 //
4 void ponto_atribui (Ponto* p, float x, float y) {
5     if (p != NULL) {
6         p->x=x;
7         p->y=y;
8     }
9 }
```

TAD – Exemplo 03

```
1  //
2  // Retorna a distancia entre dois pontos
3  //
4  float ponto_distancia (Ponto* p1, Ponto* p2) {
5          float dx, dy;
6
7          dx = p1->x - p2->x;
8          dy = p1->y - p2->y;
9          return (sqrt(dx*dx+dy*dy));
10 }
```

TAD – Exemplo 03

```
1  //
2  // move o ponto para as coordenadas (x, y)
3  //
4  void ponto_move (Ponto* p, float x, float y, int movimento) {
5
6        //
7        // Move o ponto p, a partir de sua posicao atual, para
8        // a posicao (x, y) segundo um certo tipo de movimento.
9        //
10       // Por exemplo: 1 - linear
11       // 2 - zig-zag
12       // ...
13       //
14
15 }
```

TAD – Exemplo 03

```
1 //
2 // Oculta (torna invisivel) o ponto
3 //
4 void ponto_oculta (Ponto* p) {
5
6     p->visibilidade = false;
7 }
```

TAD – Exemplo 03

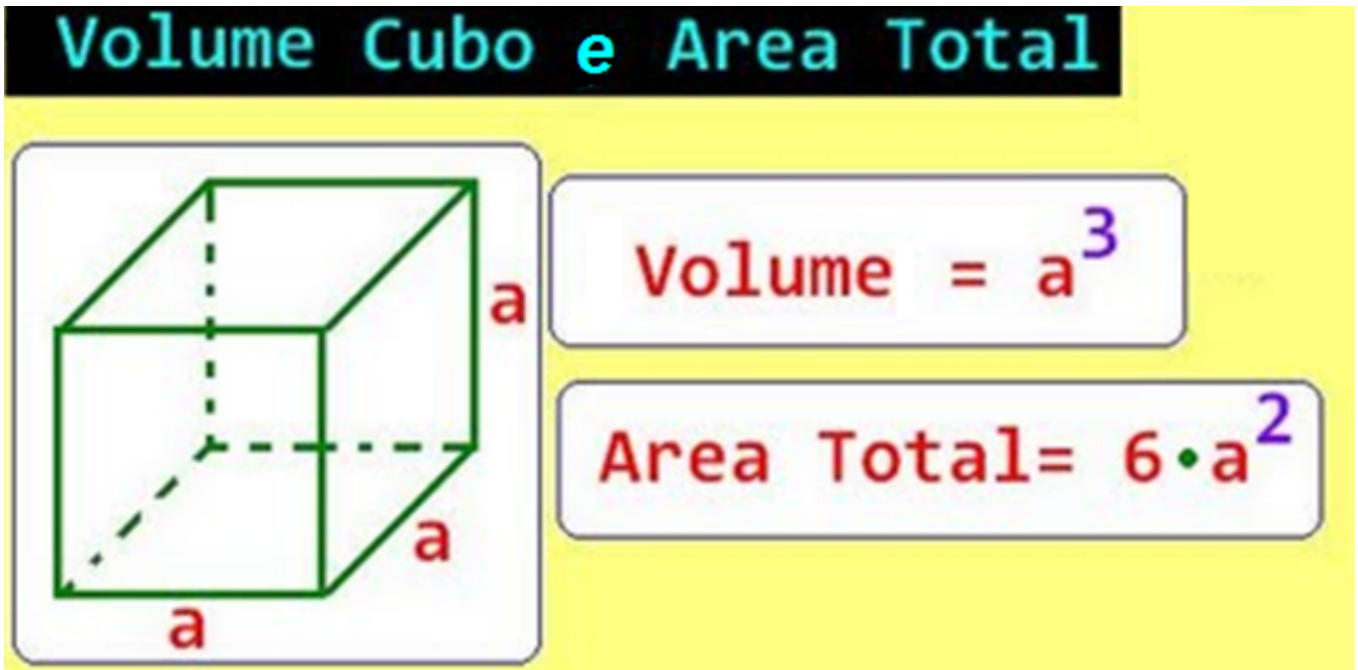
```
1 //
2 // Mostra (torna visivel) o ponto
3 //
4 void ponto_mostra (Ponto* p) {
5
6     p->visibilidade = true;
7 }
```

TAD – Exercícios

1 Conceba, planeje e implemente, em $\mathbb{C}$, um TAD para representar um cubo tridimensional.
São necessárias as seguintes operações, consideradas *fundamentais*:

(1) criar o cubo;
(2) destruir o cubo;
(3) retornar o comprimento da aresta;
(4) retornar o perímetro das arestas;
(5) retornar a área de uma face do cubo;
(6) retornar a área total das faces do cubo;
(7) retornar o volume do cubo;
(8) retornar o comprimento de suas diagonais.

TAD – Exercícios



Cubo.h

```
1  //
2  // cubo.h
3  //
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <math.h>
7
8  typedef struct cubo Cubo;
9
10 Cubo* cubo_cria(float a);
11 void cubo_libera(Cubo* c);
12 float cubo_acessa(Cubo* c);
13 void cubo_atribui(Cubo* c, float a);
14 float cubo_perimetro(Cubo* c);
15 float cubo_areaFace(Cubo* c);
16 float cubo_areaTotal(Cubo* c);
17 float cubo_volume(Cubo* c);
18 float cubo_diagonal(Cubo* c);
```

Cubo.c

```c
//
// cubo.c
//
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cubo.h"

struct cubo {
    float a;
};
```

Cubo.c

```
1  //
2  // Cria um cubo
3  //
4  Cubo* cubo_cria (float a) {
5          Cubo* c = (Cubo*) malloc(sizeof(Cubo));
6          if (c != NULL) {
7                  c->a = a;
8          }
9          return (c);
10 }
```

Cubo.c

```
1 //
2 // Libera (desaloca) o cubo
3 //
4 void cubo_libera (Cubo* c) {
5       if (c != NULL) {
6               free(c);
7           }
8 }
```

Cubo.c

```
1 //
2 // Acessa um cubo
3 //
4 float cubo_acessa (Cubo* c) {
5         return (c->a);
6 }
```

Cubo.c

```
1 //
2 // Atribui um valor a aresta do cubo
3 //
4 void cubo_atribui (Cubo* c, float a) {
5         if (c != NULL) {
6             c->a = a;
7         }
8 }
```

Cubo.c

```
1 //
2 // Retorna o perimetro do cubo
3 //
4 float cubo_perimetro (Cubo* c) {
5         if (c != NULL) {
6             return(12 * c->a);
7         }
8 }
```

Cubo.c

```
1 //
2 // Retorna a area da face do cubo
3 //
4 float cubo_areaFace (Cubo* c) {
5         if (c != NULL) {
6             return(c->a * c->a);
7         }
8 }
```

Cubo.c

```
1 //
2 // Retorna a area total das faces do cubo
3 //
4 float cubo_areaTotal (Cubo* c) {
5         if (c != NULL) {
6             return(6 * c->a * c->a);
7         }
8 }
```

Cubo.c

```
1 //
2 // Retorna o volume do cubo
3 //
4 float cubo_volume (Cubo* c) {
5         if (c != NULL) {
6             return(c->a * c->a * c->a);
7         }
8 }
```

Cubo.c

```
1 //
2 // Retorna a diagonal do cubo
3 //
4 float cubo_diagonal (Cubo* c) {
5         if (c != NULL) {
6             return(sqrt(3) * c->a);
7         }
8 }
```

Cubo.c

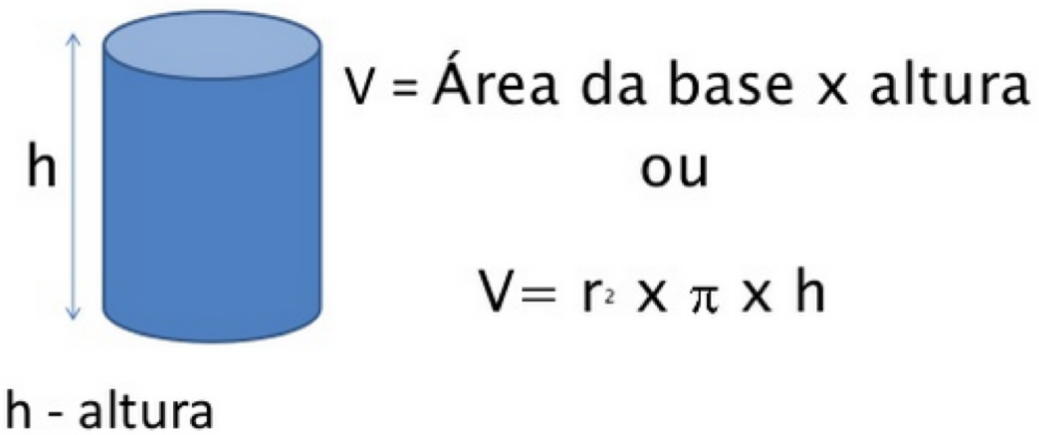
```c
//
// cubo.c -- corpo principal
//
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cubo.h"

int main(){
        float aresta;
        Cubo* variavelCubo;

        printf("digite o valor da aresta do cubo: ");
        scanf("%f",&aresta);
        variavelCubo = cubo_cria(aresta);
        printf("aresta = %.2f\n", cubo_acessa(variavelCubo));
        printf("perimetro = %.2f\n", cubo_perimetro(variavelCubo));
        printf("diagonal = %.2f\n", cubo_diagonal(variavelCubo));
        printf("area = %.2f\n", cubo_area(variavelCubo));
        printf("volume = %.2f\n", cubo_volume(variavelCubo));
        cubo_libera(variavelCubo);
        return (0);
}
```

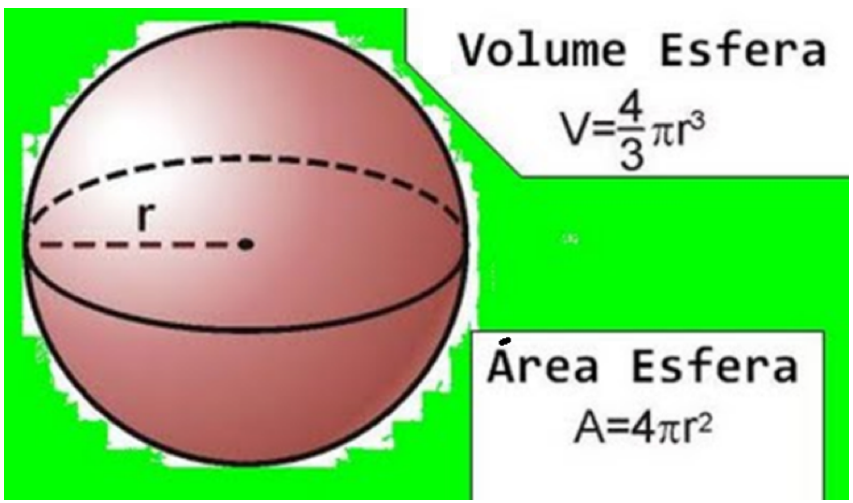
Exercícios

2 Conceba, planeje e implemente, em $\mathbb{C}$, um TAD para representar um cilindro reto.
Inclua as funções de inicialização necessárias e as operações que retornem: (a) a altura; (b) o raio; (c) a área de sua base; (d) área da face; e (e) o volume deste cilindro.



TAD – Exercícios

3 Conceba, planeje e implemente, em $\mathbb{C}$, um TAD para representar uma esfera.
Inclua as funções de inicialização necessárias e as operações que retornem: (a) o raio; (b) a área da superfície; e (c) o volume.



TAD – Exercícios

4 Conceba, planeje e implemente, em $\mathbb{C}$, um TAD para representar um *conjunto de números naturais*.
Inclua as funções de inicialização necessárias e as operações que:

(1) crie um conjunto inicialmente *vazio*;
(2) inclua um elemento num conjunto;
(3) exclua um elemento de um conjunto;
(4) teste se um elemento pertence a um conjunto;
(5) teste se o conjunto está *vazio*;
(6) retornar o maior elemento do conjunto;
(7) retornar o menor elemento do conjunto.

TAD – Exercícios

4 Conceba, planeje e implemente, em $\mathbb{C}$, um TAD para representar um *conjunto de números naturais*.
Também inclua funções para:

(1) retornar o número de elementos de um conjunto;
(2) retornar o número de elementos maior/menor que um certo valor x;
(3) comparar se dois conjuntos são idênticos;
(4) identificar se um conjunto é *subconjunto* de outro conjunto;
(5) gerar o complemento de um conjunto em relação a outro conjunto;
(6) gerar a diferença entre dois conjuntos;
(7) gerar o conjunto das partes de um conjunto;

TAD – Exercícios

5 Conceba, planeje e implemente, em $\mathbb{C}$, um TAD que represente uma *data*, concebendo:

(1) as funções de inicialização necessárias;
(2) funções convenientes para a manipulação de datas.

Referências Bibliográficas

(1) ASCÊNCIO, A. F. G. e ARAÚJO, G. S. de. *Estruturas de dados*, São Paulo: Prentice Hall, 2010;

(2) CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. e STEIN, C. *Introduction to algorithms*, 3ª edição, MIT Press, 2009;

(3) FERRARI, R.; RIBEIRO, M. X.; DIAS, R. L. e FALVO, M. *Estruturas de dados com jogos*, 1ª edição, São Paulo: Elsevier, 2014;

(4) GOODRICH, M. T. & TAMASSIA, R. *Estruturas de dados e algoritmos em Java*, 4ª edição, São Paulo: Bookman, 2007;

Referências Bibliográficas

(5) SEBESTA, Robert W., *Concepts in programming languages*, 10th. ed., Pearson, 2012;

(6) SKIENA, S. S. *The algorithm design manual*, 2^{nd} ed., London:Springer-Verlag, 2008.

(7) TANENBAUM, A. A., LANGSAM, Y., e AUGUSTEIN, M. J. *Estruturas de dados usando C*, Makron Books, 1995;

(8) ZIVIANI, N. *Projeto de algoritmos: com implementações em Pascal e C*, 2ª edição, Cengage Learning, 2009.

Saiba Mais...

Saiba Mais...

...

- vídeos indicados na área da disciplina na Plataforma Turing do INF/UFG;
- vídeos disponíveis na *web*. Por exemplo, no YouTube;
- textos explicativos, e códigos-fonte, publicados em diversos *websites* que abordam estruturas de dados.

Figura: *Sunday*, by Mark Kostabi