AED1 – Aula 4
Tipos abstratos de dados

Prof. Dr. Aldo Díaz
aldo.diaz@ufg.br

Instituto de Informática

Universidade Federal de Goiás

10/11/2022

Agenda

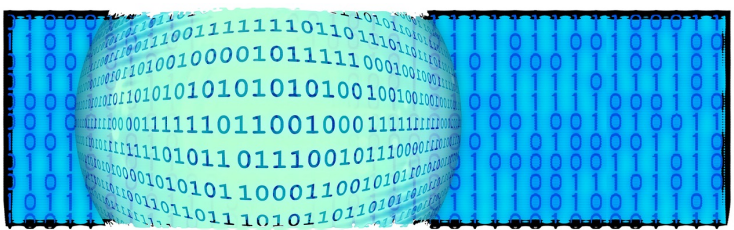
Introdução

Dado

Um *dado* representa informação sob a forma de texto, imagem, vídeo, áudio, etc.

Dados podem ser armazenados (*e.g.*, na RAM, HDD, NVM) e processados (*e.g.*, em CPU, GPU, TPU, NPU).

Internamente, são simples sequencias de *bits* (ou *qubits*).



Introdução

Tipo de dado

Conjunto de valores que uma variável pode assumir e, conjunto de operações que podem ser realizadas com ela.

- **Exemplo** – Se a variável x é `tipo inteiro`, espera-se:
  1. Que assuma valores no conjunto $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
  2. Que estejam disponíveis as operações de adição $(+)$, subtração $(-)$, multiplicação $(\times)$ e divisão inteira $(/)$.

- **Exemplo** – Se a variável é um caractere, espera-se:
  1. Valores definidos por uma *tabela de codificação* (ASCII, UNICODE, etc.).
  2. Operações de conversão maiúscula/minúscula, comparação $(<, >, \leq, \geq, \neq, \ldots)$.

Introdução

Tipo de dado – 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 línea)
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	▀	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	α	255	FFh	

Introdução

Tipo de dado

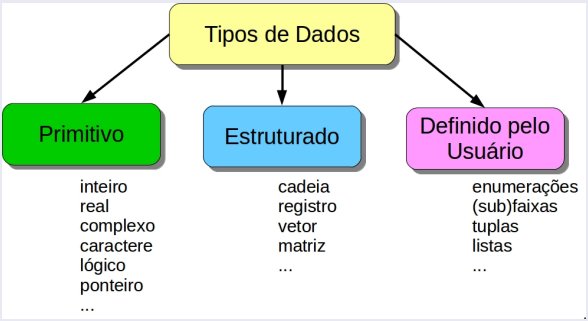
O par (`valores`, `operações`) pode variar segundo a implementação.

• **Exemplo** – Em um sistema, a implementação dos inteiros pode permitir números na faixa $[-2^{31}, 2^{31} - 1]$, enquanto em um outro, os inteiros podem estar na faixa $[-2^{63}, 2^{63} - 1]$.

Introdução

Tipo de dado – Classificação

- Primitivo
- Estruturado
- Construído (*user-defined*)



Introdução

Tipo de dado – Primitivo

- `float`
- `(unsigned) char`
- `(long) double`
- `(unsigned|short|long|long long) int`

Introdução

Tipo de dado – Estruturado (struct)

```
1  // Composicao de tipos primitvos
2  struct ContaCorrente {
3      unsigned int numero;
4      char * nomeTitular;
5      unsigned int telefoneTitular;
6      bool contaConjunta;
7      char * nomeDependente;
8      float saldoAtual;
9      bool estaAtiva;
10 }
```

Introdução

Tipo de dado – Estruturado (union)

```
1 // Variavel unica que representa multipos tipos de dados
2 union Data {
3     int intN;
4     float floatF;
5     char str[20];
6 } data;
```

Introdução

Tipo de dado – Definido pelo usuário (enum)

```c
#include <stdio.h>

// Declaracao de uma enumeracao
enum Dia {Dom, Seg, Ter, Qua, Qui, Sex, Sab};

int main() {
    enum Dia dia = Qui;

    if(dia == Dom)
        printf("FDS\n");
    else
        printf("%d\n", dia);

    return(0);
}
```

Introdução

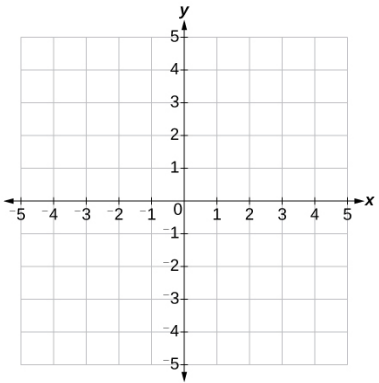
Tipo de dado – Definido pelo usuário (bool)

```c
#include <stdio.h>
#include <stdbool.h>

int main() {
    bool emCrash = true; // Tipo logico (boolean)

    if(!emCrash)
        printf("Sistema OK!\n");
    else
        printf("Sistema em CRASH.\n");

    return (0);
}
```

Tipos abstratos de dados

Um TAD é um tipo de dado criado cujas operações podem ser especificadas.

• **Exemplo** – Em *Computação Gráfica* deseja-se um TAD que represente um *ponto 2D* (bidimensional).



Exemplo

Ponto 2D

Um *ponto 2D* possui:

1. Valores – Dupla ordenada `Ponto(x,y)` formada por dois reais:
   - `x` ("abscissa") – Um número real, $x \in \mathbb{R}$
   - `y` ("ordenada") – Um número real, $y \in \mathbb{R}$
2. Operações – Aplicáveis sobre o tipo `Ponto`. Quais são?

Exemplo

Ponto 2D – Operações

- `ponto_cria`: Aloca memória para um ponto.
- `ponto_libera`: Libera a memória alocada por um ponto.
- `ponto_acessa`: Retorna as coordenadas de um ponto.
- `ponto_atribui`: Atribui valores às coordenadas de um ponto.
- `ponto_distancia`: A distância Euclidiana entre dois pontos.
- `ponto_move`: Move um ponto de uma posição para outra.
- `ponto_oculta`: Torna o ponto invisível.
- `ponto_mostra`: Torna o ponto visível.

Exemplo

ponto – Definição do TAD

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <math.h>
4
5  typedef struct ponto Ponto;
6
7  struct ponto {
8      float x;
9      float y;
10     bool visibilidade; // true = visivel, false = invisivel
11 };
```

Exemplo

ponto – Definição do TAD

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

```
1  void ponto_libera(Ponto* p) {
2      if(p != NULL)
3          free(p);
4  }
```

Exemplo

ponto – Definição do TAD

```
1 void ponto_acessa(Ponto* p, float* x, float* y) {
2     if(p != NULL) {
3         *x = p->x;
4         *y = p->y;
5     }
6 }
```

```
1 void ponto_atribui(Ponto* p, float x, float y) {
2     if(p != NULL) {
3         p->x = x;
4         p->y = y;
5     }
6 }
```

Exemplo

ponto – Definição do TAD

```
1  float ponto_distancia(Ponto* p1, Ponto* p2) {
2      float dx, dy;
3
4      dx = p1->x - p2->x;
5      dy = p1->y - p2->y;
6
7      return(sqrt(dx*dx + dy*dy));
8  }
```

```
1  void ponto_move(Ponto* p, float x, float y, int movimento) {
2      /* Move o ponto p, para a posicao (x, y) segundo o tipo de movimento:
3              1 - Linear
4              2 - Zig-zag
5      */ ...
6  }
```

Exemplo

ponto – Definição do TAD

```
1 void ponto_oculta(Ponto* p) {
2     p->visibilidade = false;
3 }
```

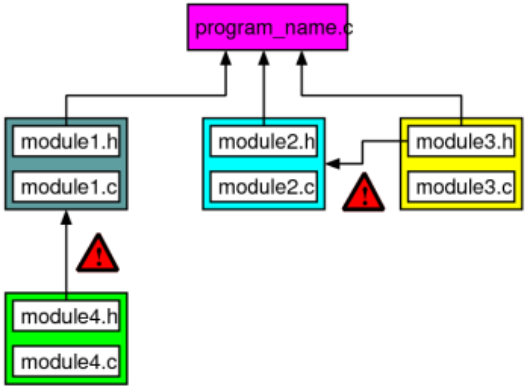
```
1 void ponto_mostra(Ponto* p) {
2     p->visibilidade = true;
3 }
```

Exemplo

ponto – Definição do TAD

```
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     ponto_acessa(p, &xp, &yp);
12     ponto_acessa(q, &xq, &yq);
13     printf("Distancia entre os pontos: %.1f\n", ponto_distancia(p, q));
14     ponto_libera(p);
15     ponto_libera(q);
16
17     return(0);
18 }
```

Modularização



Modularização

Para criar um TAD em `C`, convenciona-se preparar 2 arquivos:

- `<arquivo.h>` – Contém os *protótipos* de:
  - Funções
  - Ponteiros
  - Variáveis globais
- `<arquivo.c>` – Contém:
  - A declaração do tipo de dados
  - Implementação das operações (funções)

Modularização

Ponto 2D – Exemplo

Vamos redefinir "ponto 2D" utilizando *modularização*:

1. Arquivo `ponto.h`
   - Variáveis globais
   - Protótipos das funções
   - Ponteiros
2. Arquivo `ponto.c`
   - Implementação das operações (funções)

Modularização

ponto.h

```
1  // Variaveis globais
2  typedef struct ponto Ponto;
3
4  // Funcoes
5  Ponto* ponto_cria(float x, float y, bool visibilidade);
6  void ponto_libera(Ponto* p);
7  void ponto_acessa(Ponto* p, float* x, float* y);
8  void ponto_atribui(Ponto* p, float x, float y);
9  float ponto_distancia(Ponto* p1, Ponto* p2);
10 void ponto_move(Ponto* p, float x, float y);
11 void ponto_oculta(Ponto* p);
12 void ponto_mostra(Ponto* p);
```

Modularização

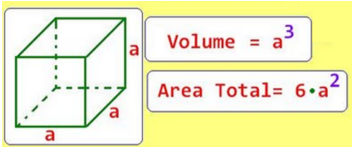
ponto.c

```c
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include "ponto.h" // Arquivo de cabecalho

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

// <<< Anexar aqui a implementacao das funcoes >>>
```

Exercício 1

Projete um TAD para representar um cubo tridimensional. Considere as seguintes operações fundamentais:

a. Criar cubo
b. Destruir cubo
c. Comprimento da aresta
d. Perímetro das arestas
e. Área de uma face
f. Área total
g. Volume do cubo
h. Comprimento das diagonais



Exercício 1

cubo.h

```
1  typedef struct cubo Cubo;
2
3  Cubo* cubo_cria(float a);
4  void cubo_libera(Cubo* c);
5  float cubo_acessa(Cubo* c);
6  void cubo_atribui(Cubo* c, float a);
7  float cubo_perimetro(Cubo* c);
8  float cubo_areaFace(Cubo* c);
9  float cubo_areaTotal(Cubo* c);
10 float cubo_volume(Cubo* c);
11 float cubo_diagonal(Cubo* c);
```

Exercício 1

cubo.c

```
#include <stdio.h>
#include <math.h>
#include "cubo.h"

struct cubo {
    float a;
};

Cubo* cubo_cria(float a) {
    Cubo* c = (Cubo*) malloc(sizeof(Cubo));
    if(c != NULL)
        c->a = a;

    return(c);
}
```

Exercício 1

cubo.c

```
1  void cubo_libera(Cubo* c) {
2      if(c != NULL)
3          free(c);
4  }
```

```
1  float cubo_acessa(Cubo* c) {
2      return(c->a);
3  }
```

```
1  void cubo_atribui(Cubo* c, float a) {
2      if(c != NULL)
3          c->a = a;
4  }
```

Exercício 1

cubo.c

```
1  float cubo_perimetro(Cubo* c) {
2      if(c != NULL)
3          return(12 * c->a);
4  }
```

```
1  float cubo_areaFace(Cubo* c) {
2      if(c != NULL)
3          return(c->a * c->a);
4  }
```

```
1  float cubo_areaTotal(Cubo* c) {
2      if(c != NULL)
3          return(6 * c->a * c->a);
4  }
```

Exercício 1

cubo.c

```
1 float cubo_volume(Cubo* c) {
2     if(c != NULL)
3         return(c->a * c->a * c->a);
4 }
```

```
1 float cubo_diagonal(Cubo* c) {
2     if(c != NULL)
3         return(sqrt(3) * c->a);
4 }
```

Exercício 1

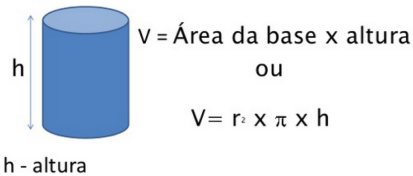
main.c

```c
#include <stdio.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);
}
```

Exercício 2

Projete um TAD para representar um cilindro reto. Inclua as funções de inicialização necessárias e as operações que retornem sua:

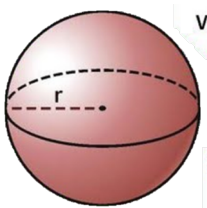
a. Altura
b. Raio
c. Área da base
d. Área da face
e. Volume deste cilindro



Exercício 3

Projete um TAD para representar uma esfera. Inclua as funções de inicialização necessárias e as operações que retornem seu:

a. Raio
b. Área da superfície
c. Volume



Volume Esfera

$$V=\frac{4}{3}\pi r^3$$

Área Esfera

$$A=4\pi r^2$$

Exercício 4

Projete um TAD para representar um *conjunto de números naturais*. Inclua as funções de inicialização necessárias e as operações que:

a. Crie um conjunto inicialmente *vazio*
b. Inclua um elemento num conjunto
c. Exclua um elemento de um conjunto
d. Teste se um elemento pertence a um conjunto
e. Teste se o conjunto está *vazio*
f. Retornar o maior elemento do conjunto
g. Retornar o menor elemento do conjunto

Exercício 4

Também inclua funções para:

h. Retornar o número de elementos de um conjunto
i. Retornar o número de elementos maior/menor que um certo valor x
j. Comparar se dois conjuntos são idênticos
k. Identificar se um conjunto é *subconjunto* de outro conjunto
l. Gerar o complemento de um conjunto em relação a outro conjunto
m. Gerar a diferença entre dois conjuntos
n. Gerar o conjunto das partes de um conjunto

Exercício 5

Projete um TAD que represente uma *data*, concebendo:

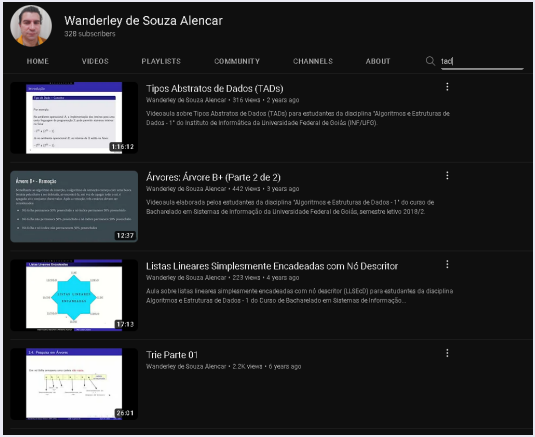
a. As funções de inicialização necessárias

b. Funções convenientes para a manipulação de datas

Saiba mais...

youtu.be

- Canal do Prof. Wanderley de Souza Alencar (INF/UFG)



Referências

1. ASCÊNCIO, A. F. G. e ARAÚJO, G. S. de. *Estruturas de dados*, São Paulo: Prentice Hall, 2010.
2. FERRARI, R.; RIBEIRO, M. X.; DIAS, R. L. e FALVO, M. *Estruturas de dados com jogos*, 1ª edição, São Paulo: Elsevier, 2014.
3. SKIENA, S. S. *The algorithm design manual*, 3 Ed. London:Springer-Verlag, 2020.