



UNIVERSIDADE  
FEDERAL DO CEARÁ



# Programação

## (CK0226 – 2022.2)

Curso: Ciência da Computação

Professor: Lincoln Souza Rocha

E-mail: [lincoln@dc.ufc.br](mailto:lincoln@dc.ufc.br)

# Introdução à Linguagem C – Parte IV

# Sumário

- Vetores
- Matrizes
- Cadeia de Caracteres

# Vetores

Um vetor é uma estrutura de dados que define um conjunto enumerável armazenado sequencialmente na memória.

```
int  vetor[10];
```

declaração de vetor com 10 posições

```
int  vetor[10] = {1,2,3,4,5,6,7,8,9,10};
```

declaração/inicialização de vetor com 10 posições

```
int  vetor[] = {1,2,3,4,5,6,7,8,9,10};
```

declaração/inicialização de vetor com 10 posições

```
vetor[0] = 1;
```

```
vetor[1] = 2;
```

```
vetor[2] = 3;
```

```
(...)
```

```
vetor[6] = 7;
```

```
vetor[7] = 8;
```

```
vetor[8] = 9;
```

atribuição indexada de valores ao vetor

```
vetor[10] = 11;
```

atribuição incorreta (invasão de memória)

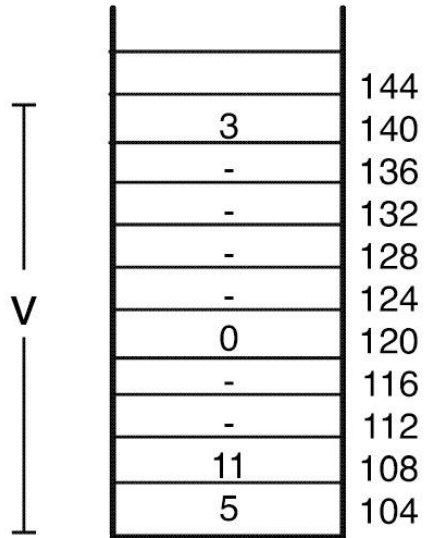
# Vetores

```
int v [10];
```

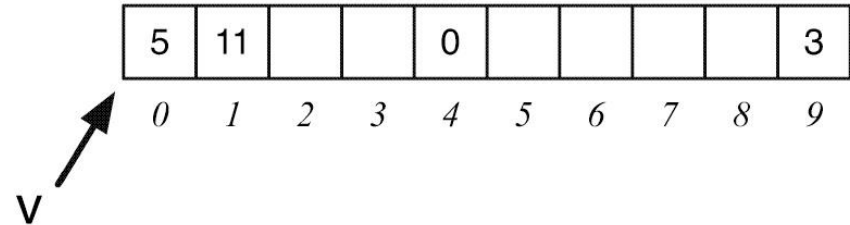
```
v[0] = 5;  
v[1] = 11;  
v[4] = 0;  
v[9] = 3;
```

## Nota!

O espaço de memória ocupado por um vetor de 10 posições é:  
➤ 10 x 4 bytes = 40 bytes



(a)



(b)

# Vetores e Ponteiros

No nome da variável vetor aponta para o endereço do primeiro espaço de memória do vetor. C permite aritmética de ponteiros.

$v+0$ : é o primeiro elemento de  $v$   
 $v+1$ : é o segundo elemento de  $v$   
 $v+3$ : é o terceiro elemento de  $v$   
(...)  
 $v+9$ : é o último elemento de  $v$

## Nota!

- $\&v[i]$  é equivalente a  $(v+i)$
- $*(v+i)$  é equivalente a  $v[i]$

	$v$	
$(v+9) \rightarrow$		140
$(v+8) \rightarrow$		136
$(v+7) \rightarrow$		132
$(v+6) \rightarrow$		128
$(v+5) \rightarrow$		124
$(v+4) \rightarrow$		120
$(v+3) \rightarrow$		116
$(v+2) \rightarrow$		112
$(v+1) \rightarrow$		108
$(v+0) \rightarrow$		104

## Exercício @ Classe

Implemente um programa que faz o cálculo da média ( $m$ ) e da variância ( $v$ ) de uma amostra de tamanho  $n$ . Conforme as fórmulas abaixo:

$$m = \frac{\sum_{i=1}^n x_i}{n}, v = \frac{\sum_{i=1}^n (x_i - m)^2}{n}$$

# Vetores: Média e Variância

```
#include <stdio.h>
# include <stdlib.h>
#define N 100 /* dimensão do vetor */

int main (void ) {
    int n; /* número de valores */
    float x[N]; /* vetor dos valores */

    printf("Entre com o numero de valores: ");
    scanf("%d", &n);

    if (n > N) {
        printf("Valor ultrapassa o limite de % d.\ n", N);
        return 1;
    }

    printf("Entre com os valores:\ n");

    for (int i=0; i<n; ++i) {
        scanf("%f", &x[i]);
    }

    (...)
```



# Vetores: Média e Variância

```
(...)  
  
float m = 0.0f;  
for (int i=0; i<n; ++i) {  
    m += x[i];  
}  
  
m /= n;  
  
float v = 0.0f;  
for (int i=0; i<n; ++i) {  
    v += (x[i]-m) * (x[i]-m);  
}  
  
v /= n;  
  
printf("Media: %f\ nVariancia: %f\ n", m, v);  
  
return 0;  
}
```

# Passagem de Vetor para Função

Consiste em passar o endereço da primeira posição do vetor. A função deve ter parâmetro do tipo ponteiro para armazenar valor. Nesse caso, “passar um vetor para uma função” é equivalente a “passar o endereço inicial do vetor”. Os elementos do vetor não são copiados para a função, apenas o endereço do primeiro elemento.

A função pode alterar os valores dos elementos do vetor pois recebe o endereço do primeiro elemento do vetor (e não uma cópia dos elementos do vetor propriamente dito).

# Vetores: Média e Variância (v2)

```
#include <stdio.h>
#include <stdlib.h>
#define N 100 /* dimensão do vetor */

void captura (int n, float* x);
float media (int n, float* x);
float variancia (int n, float* x, float m);

int main (void ) {
    int n; /* número de valores */
    float x[N]; /* vetor dos valores */

    printf("Entre com o numero de valores: ");
    scanf("% d", & n);

    if (n > N) {
        printf("Valor ultrapassa o limite de % d.\ n", N);
        return 1;
    }
    captura(n, x);
    float m = media(n, x);
    float v = variancia(n, x, m);
    printf("Media: %f\ nVariancia: %f\n", m, v);
    return 0;
}
(...)
```

# Vetores: Média e Variância (v2)

(...)

```
void captura (int n, float* x) {  
    printf("Entre com os valores:\ n");  
    for (int i=0; i<n; ++i)  
        scanf("% f", & x[i]);  
}
```

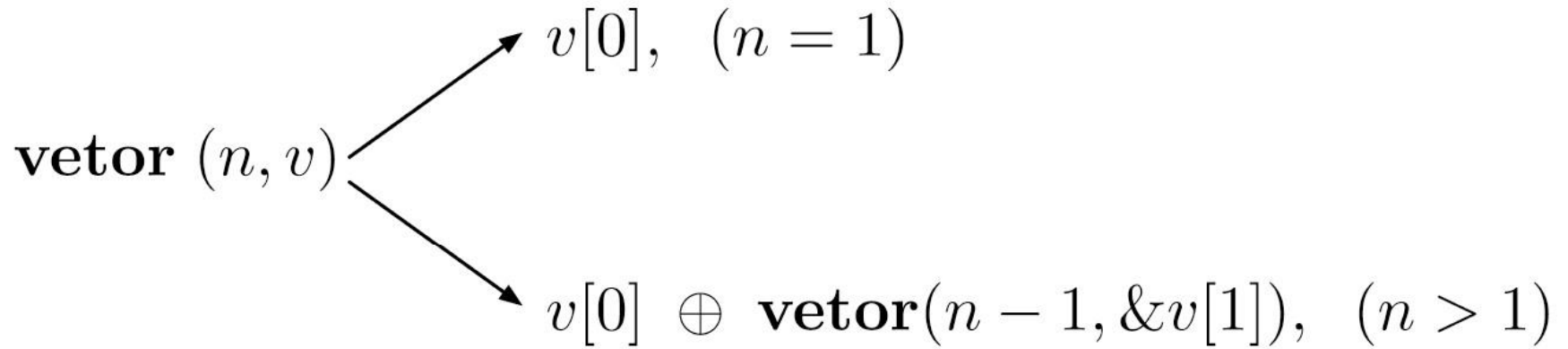
```
float media (int n, float* x) {  
    float m = 0.0 f;  
    for (int i=0; i<n; ++i)  
        m += x[i];  
    return m / n;  
}
```

```
float variancia (int n, float* x, float m) {  
    float v = 0.0 f;  
    for (int i=0; i<n; ++i)  
        v += (x[i]-m) * (x[i]-m);  
    return v / n;  
}
```

# Recursão com Vetores

```
float maximo (int n, float* v) {  
    float m = v[0]; /* armazena valor máximo */  
    int i;  
    for (i=1; i<n; ++i) {  
        if (v[i] > m)  
            m = v[i];  
    }  
    return m;  
}
```

# Recursão com Vetores



# Recursão com Vetores

```
float maximo (int n, float* x) {  
    if (n == 1) {  
        return x[0];  
    } else {  
        float msub = maximo(n-1, &x[1]);  
        return x[0] > msub ? x[0] : msub;  
    }  
}
```

## Exercício @ Classe

Avaliar um polinômio significa avaliar o valor numérico do polinômio,  $y = a(x)$ , para determinado  $x$ . Implemente um programa que calcule o valor numérico de um polinômio usando a fórmula abaixo.

$$y = \sum_{i=0}^g a_i x^i$$



# Matrizes Bidimensionais

São uma estrutura de dados que definem um conjunto bidimensional enumerável armazenado na memória.

```
float mat[3][4];
```

**declaração de matriz 3x4 posições**

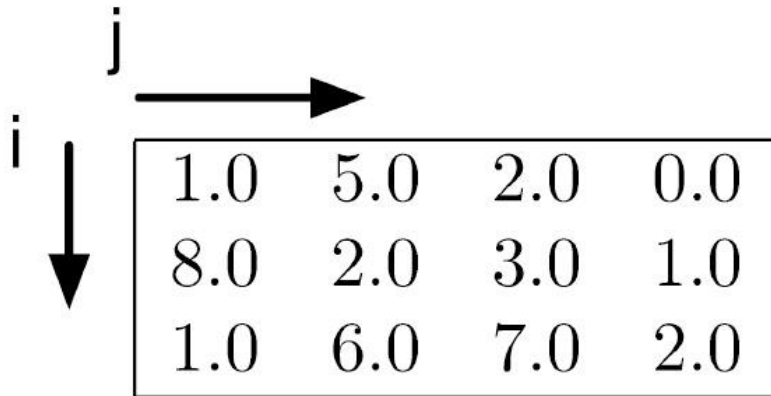
```
float mat [3][4] = {  
    {1.0f, 5.0f, 2.0f, 0.0f},  
    {8.0f, 2.0f, 3.0f, 1.0f},  
    {1.0f, 6.0f, 7.0f, 2.0f}  
};
```

**declaração/inicialização de matriz 3x4 posições**

```
mat [0][0] = 1.0f;  
mat [0][1] = 5.0f;  
mat [0][2] = 2.0f;  
mat [0][3] = 0.0f;
```

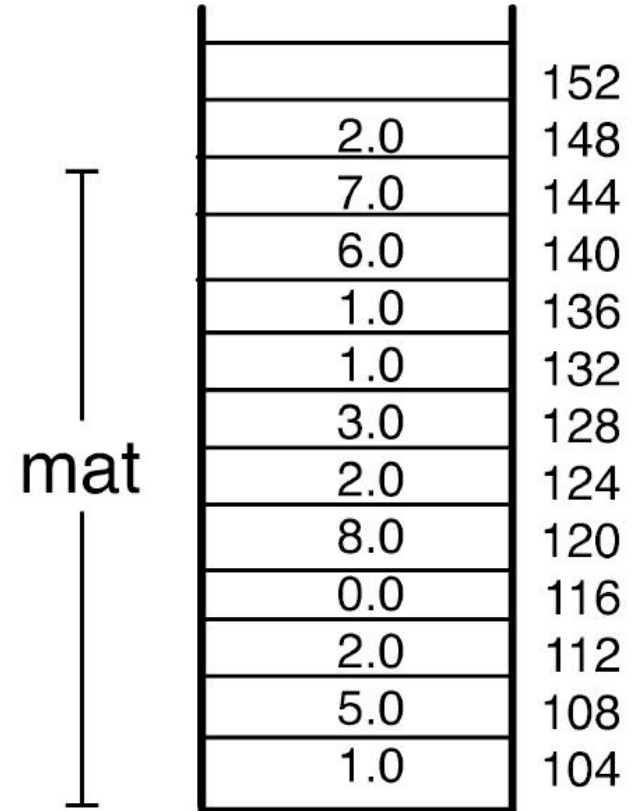
**atribuição indexada de valores à matriz**

# Matrizes Bidimensionais



A 3x4 matrix is shown with row index  $i$  pointing downwards and column index  $j$  pointing to the right.

1.0	5.0	2.0	0.0
8.0	2.0	3.0	1.0
1.0	6.0	7.0	2.0



A 1D array named `mat` is shown, containing 12 elements. The memory address for each element is listed to the right of the array.

	152
2.0	148
7.0	144
6.0	140
1.0	136
1.0	132
3.0	128
2.0	124
8.0	120
0.0	116
2.0	112
5.0	108
1.0	104

# Percorrendo Matrizes

```
#include <stdio.h>

int main (void ) {
    float mat [3][4] = {
        {1.0f, 5.0f, 2.0f, 0.0 f},
        {8.0f, 2.0f, 3.0f, 1.0 f},
        {1.0f, 6.0f, 7.0f, 2.0 f}
    };

    printf("\n");
    for (int i=0; i<3; i++) {
        for (int j=0; j<4; j++) {
            printf("M(%d,%d)=%.1f \t", i, j, mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

M (0 ,0)=1.0

M (0 ,1)=5.0

M (0 ,2)=2.0

M (0 ,3)=0.0

M (1 ,0)=8.0

M (1 ,1)=2.0

M (1 ,2)=3.0

M (1 ,3)=1.0

M (2 ,0)=1.0

M (2 ,1)=6.0

M (2 ,2)=7.0

M (2 ,3)=2.0

# Passagem de Matrizes para Funções

```
#include <stdio.h>

void imprime(int m, float mat[][4]); /* OU void imprime(int m, float (* mat)[4]) */

int main (void ) {
    float mat [3][4] = {
        {1.0f, 5.0f, 2.0f, 0.0 f},
        {8.0f, 2.0f, 3.0f, 1.0 f},
        {1.0f, 6.0f, 7.0f, 2.0 f}
    };
    imprime(3, mat);
    return 0;
}

void imprime(int m, float mat[][4]) {
    printf("\n");
    for (int i=0; i<m; i++) {
        for (int j=0; j<4; j++) {
            printf("M(%d,%d)=%.1 f \t", i, j, mat[i][j]);
        }
        printf("\n");
    }
}
```

## Exercício @ Classe

Uma matriz quadrada,  $M$ , é dita simétrica se  $M_{i,j} = M_{j,i}$  para qualquer elemento da matriz. Isto é, elementos em lados opostos da diagonal principal da matriz devem ser iguais. Implemente uma função que verifique se uma dada matriz quadrada  $M$  é simétrica ou não.

## Exercício @ Classe

Se  $M$  é uma matriz, sua transposta é definida por:  $T_{j,i} = M_{i,j}$ . Implemente uma função que calcule a transposta de uma dada matriz quadrada  $M$ .

# Caracteres

Representados em C pelo tipo **char**

Tamanho de char = 1 byte = 8 bits = 256 valores distintos

Tabela de símbolos (códigos). Define correspondência entre caracteres e códigos numéricos. Exemplo: ASCII

Alguns alfabetos precisam de maior representatividade (e.g., o alfabeto chinês tem mais de 256 caracteres)

# Tabela ASCII

	0	1	2	3	4	5	6	7	8	9
30			sp	!	"	#	\$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	'	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{		}	~			

Códigos ASCII de alguns caracteres que podem ser impressos (sp representa espaço em branco).



# Tabela ASCII

0	<i>null</i> : nulo
7	<i>bell</i> : campainha
8	<i>backspace</i> : volta e apaga um caractere
9	<i>tab</i> : tabulação horizontal
10	<i>newline</i> ou <i>line feed</i> : muda de linha
13	<i>carriage return</i> : volta ao início da linha
127	<i>delete</i> : apaga um caractere

Códigos ASCII de alguns caracteres de controle.

# Caracteres

Constantes de caractere são envolvidos por aspas simples:

- 'a' representa uma constante de caractere
- 'a' resulta no valor numérico associado ao caractere a

```
char c = 'a';  
printf("%d %c\n", c, c);
```

O `printf` imprime o conteúdo de `c` usando dois formatos:

- com o formato para inteiro, `%d`, imprime 97
- com o formato de caractere, `%c`, imprime a (código 97 em ASCII)

# Cadeia de Caracteres

Vetor do tipo char, terminado pelo caractere nulo ('\0'). É necessário reservar uma posição adicional no vetor para o caractere de fim da cadeia.

Uma função para manipular cadeias de caracteres recebe como parâmetro um vetor de char, processa caractere por caractere até encontrar o caractere nulo, sinalizando o final da cadeia.

# Cadeia de Caracteres

## Inicialização de cadeias de caracteres:

- Caracteres entre aspas duplas
- Caractere nulo é representado implicitamente

```
int main ( void ) {  
    char cidade[ ] = "Ipu";  
    printf("%s \n", cidade);  
    return 0;  
}
```



```
int main ( void ) {  
    char cidade[ ] = {'I', 'p', 'u', '\\0'};  
    printf("%s \n", cidade);  
    return 0;  
}
```

# Leitura de Caracteres

A leitura de caracteres e cadeias de caracteres pode ser feita através da função `scanf` usando os especificadores de formato para definirem o comportamento da função.

O `scanf` com o especificador de formato `%c` lê o valor de um único caractere fornecido via teclado.

```
char a;  
scanf ("%c", &a);
```



UNIVERSIDADE  
FEDERAL DO CEARÁ



# Programação

## (CK0226 – 2022.2)

Curso: Ciência da Computação

Professor: Lincoln Souza Rocha

E-mail: [lincoln@dc.ufc.br](mailto:lincoln@dc.ufc.br)